Facultat d'informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Convenio de cooperación educativa modalidad B
everis an NTT DATA Company
Departamento de "Digital Channels (Unidad de Tecnología)"

Herramienta de pruebas automatizadas y reporting por diferentes canales

Memoria
Defensa: 30/06/2016

Trabajo Final de Grado
Ingeniería Informática

Autor

Gabriel Carrillo López

Especialidad: Sistemas de la Información

Ponente

Enric Mayol Sarroca

Enginyeria de Serveis i Sistemes d'Informació

Resumen

En la actualidad, muchas empresas tecnológicas llevan a cabo el desarrollo y mantenimiento de portales web para grandes sistemas corporativos. En función del estado de los proyectos, el producto suele pasar por distintas entornos bien diferenciadas: local (L), integración (I), pre-producción (PP) y producción (P).

El problema vienen dado al pasar el producto de un entorno a otro, es decir, de L - I, de I - PP y de PP - P. Esto es así porque hay que probar que tanto las nuevas funcionalidades desarrolladas como todas las anteriores sigan funcionando como se espera y se deben generar los informes correspondientes. A día de hoy, aún hay proyectos en los que se sigue haciendo a mano. Esto es una tarea repetitiva (se ha de comprobar lo mismo al pasar de un entorno a otro) y presumible de contener errores que posteriormente deberán corregirse.

El propósito de este proyecto consiste en crear una herramienta de testing de portales para obtener resultados y generar informes automáticamente, reduciendo así el tiempo invertido en la realización de las pruebas y el trabajo repetitivo, aumentando la fiabilidad de las pruebas y posibilitando su reutilización.

Abstract

Nowadays, many technology companies carry out the development and maintenance of web portals for large corporate systems. Depending on the status of the projects, the product passes through several distinct environments: Local (L), integration (I), pre-production (PP) and production (P).

The problem appears when passing the product from one environment to another, ie, L - I, I - PP and PP - P. This is so because both, new developed features and the previous ones, have to be tested in order to prove its behaviour remains as expected and reports have to be generated. Today, there are still projects that are done by hand. This is a repetitive task (it checks the same when passing from one environment to another) and presumed to contain errors to be corrected later.

The purpose of this project is to create a portal testing tool for results and to generate reports automatically, thus reducing the time spent doing tests and repetitive work, increasing the reliability of the tests and enabling its reuse.

Resum

En l'actualitat, moltes empreses tecnològiques duen a terme el desenvolupament i manteniment de portals web per a grans sistemes corporatius. En funció de l'estat dels projectes, el producte sol passar per diferents entorns ben diferenciades: local (L), integració (I), pre-producció (PP) i producció (P).

El problema vé donat en passar el producte d'un entorn a un altre, és a dir, de L - I, d'I - PP i de PP - P. Això és així perquè cal provar que tant les noves funcionalitats desenvolupades com totes les anteriors segueixin funcionant com s'espera i s'han de generar els informes corresponents. Avui dia, encara hi ha projectes en els quals se segueix fent a mà. Això és una tasca repetitiva (s'ha de comprovar el mateix en passar d'un entorn a un altre) i presumible de contenir errors que posteriorment hauran de corregir-se.

El propòsit d'aquest projecte consisteix a crear una eina de testing de portals per obtenir resultats i generar informes automàticament, reduint així el temps invertit en la realització de les proves i el treball repetitiu, augmentant la fiabilitat de les proves i possibilitant la seva reutilització.

Agradecimientos

Quiero dar las gracias al ponente, Enric Mayol, por su objetividad, su habilidad para resolver problemas y su alta disponibilidad para ayudar siempre con una sonrisa de oreja a oreja, tanto en temas profesionales como personales.

También quiero agradecer a mi director Raúl Bori la oportunidad de incorporarme en su equipo y hacer este proyecto, sus revisiones y punto de vista como profesional del sector y su gran liderazgo. Sin ti, el "Core team" no existiría.

A mi compañero Jose, por su eterna ayuda y paciencia, sus explicaciones y aperitivos sin los que al proyecto le hubiese faltado el toque "picante".

A la línea de Digital Channels, por el buen ambiente de trabajo diario, aliñadas con un equilibrio perfecto de bromas, seriedad, retos y compañerismo.

A Cristina, por su apoyo incondicional en todo lo que hago. Sin tus ánimos, cariño y paciencia se hubiese hecho mucho más difícil.

A todos mis compañeros de la universidad y de fuera que me han ayudado y aconsejado en los puntos críticos del camino, al principio a escoger asignaturas, después especialidad, y más adelante empresa y proyecto: Juanan, Mercè, Moles, Maikals, David y un largo etc.

Índice

1. Introducción

"*Los sistemas software son cada vez más grandes y más complejos y los clientes demandan software más confiable que sea desarrollado más rápidamente*" [Som'05]. Esta cita induce a pensar que las pruebas de software tienen cada día más importancia porque garantizan la satisfacción de los requisitos. Sin embargo, cuando éstas se automatizan, además de garantizar la satisfacción de los requisitos del software, se garantiza su calidad y se ahorra tiempo y recursos en su desarrollo al localizar el mayor número de deficiencias en el menor tiempo.

La comprobación del software consiste en la *verificación dinámica*[1] del comportamiento de un programa en un conjunto finito (número limitado) de casos de prueba (no se ejecuta de cualquier forma).

Los casos de prueba permiten tanto verificar el comportamiento esperado de una aplicación, como detectar errores. Interesa que con el menor número de casos se encuentren el mayor número de errores. Para ello se necesitan unos buenos casos de prueba, que se caracterizan por:

- Tener una alta probabilidad de encontrar un fallo (cubrir el máximo de posibilidades).
- No ser redundante (si funciona, no se vuelve a probar).
- Ser la mejor entre las suyas (si se dispone de varias opciones, se escoge la mejor).
- Tener una complejidad media (muy sencilla no aporta y muy compleja dificultará saber lo que falla).

Hay que matizar que un programa que pasa todos sus casos de prueba no es un programa sin errores, puesto que una prueba solo puede hallar los errores que busca. **De aquí la importancia de planear un buen diseño de pruebas**.

[1] Implica siempre la ejecución del programa

2. Contextualización

El proyecto "Herramienta de pruebas automatizadas y reporting por diferentes canales" se enmarca dentro de la especialidad de Sistemas de la Información del plan de estudios del Grado en Ingeniería Informática de la Facultad de Informática de Barcelona (FIB) - Universidad Politécnica de Cataluña (UPC) - BarcelonaTech. Este proyecto se ha llevado a cabo en el departamento de "Digital Channels (Unidad de Tecnología)" de la empresa "everis an NTT DATA Company" [Eve96], una consultora multinacional que ofrece soluciones de negocio, estrategia, desarrollo y mantenimiento de aplicaciones tecnológicas y outsourcing.

En este departamento se desarrollan y mantienen portales de diversa índole (conocimiento, intranets, contenidos, capa social) sobre Liferay [Lif00], en PHP [PHP95], con Sharepoint [Sha13] u Oracle WebCenter Sites [OWS14] para grandes sistemas corporativos. La capa social de un sistema corporativo permite añadir una serie de funcionalidades sociales a la tecnología ya existente en la entidad, como por ejemplo:

- Microblogging: entornos de colaboración social.
- Noticias.
- Integración entre sistemas (novedades sobre los usuarios) mediante un sistema de seguimiento.
- Mejora de la usabilidad en la gestión de grupos de trabajo.
- Administración sencilla de usuarios y contactos.
- Acceso vía móvil a la plataforma.

Estos portales están desplegados en 3 ó 4 entornos distintos:

- producción (PRO).
- pre-producción (PRE).
- integración (opcional).
- local.

El entorno de producción es un servidor web al que acceden las peticiones de los usuarios (el servicio que se ofrece). El entorno de pre-producción suele ser una copia del de producción, pero al que solo tienen acceso los clientes para hacer las pruebas del código encargado. El entorno de integración es el servidor en el que los desarrolladores *suben*[2] los nuevos desarrollos junto al resto del proyecto (ver que todo funciona). El entorno local es la máquina personal de cada desarrollador, es decir, su ordenador. Es donde cada uno lleva a cabo su trabajo: desarrollar, programar, "picar código".

Cuando un programador ha desarrollado algo, lo *sube* a integración para corroborar que todo funciona como es debido. Aquí, todos los integrantes del equipo tendrán acceso al código *subido*. En el caso de que no exista el entorno de integración, los desarrolladores hacen todo lo descrito directamente en PRE. Cada cierto tiempo (acordado con el cliente), se *suben* todos los cambios hechos en integración a PRE. De este modo el cliente ve y corrobora los cambios llevados a cabo.
Cuando el cliente ha revisado los cambios de PRE (suele ser un espacio corto de tiempo - horas/días), da el visto bueno y se suben los cambios hechos en PRE a PRO.

[2] Cargarlo en el servidor

2.1. Actores implicados

El proyecto va dirigido a los desarrolladores de portales web interesados en ver el progreso del testeo de la versión de una funcionalidad, lo que está fallando, los errores o la necesidad de cambios.

Para poder crear un buen conjunto de pruebas de un sistema, el conjunto de las partes debe poseer una serie de habilidades como el ojo crítico, la curiosidad, la atención al detalle, la buena comunicación y la experiencia, entre otras.

Beatriz Pérez [Per06] distingue dos roles principales en el proceso de testeo: testers y líderes. La función del líder es, según [Per06] la de planificar, monitorizar y controlar las actividades y tareas de las pruebas. Las funciones del tester, sin embargo, afirma que varían en función del papel de la persona que adquiera el rol:

- un desarrollador está a nivel de componente e integración (foco del proyecto).
- un experto de negocio está a nivel de prueba de aceptación.
- un usuario está a nivel de prueba de aceptación operacional.

En el caso que nos ocupa, el líder del proceso de testeo es el líder del equipo de la empresa en la que se ha desarrollado el proyecto. La función del tester a nivel de componente e integración será la que se llevará a cabo en este proyecto. La función del tester a nivel de pruebas de aceptación y de pruebas de aceptación operacional será la que llevará la empresa cuando el proyecto haya finalizado y quieran usarlo como expertos de negocio y como usuarios. El prototipo de equipo que puede sacar provecho al proyecto es aquel que quiere evitar errores humanos, ejecutar las pruebas con eficiencia y generar informes automatizados del Plan de Pruebas.

Al tratarse de un TFG, encontramos también el rol del Ponente; profesor de la especialidad que da soporte durante la elaboración del proyecto y valida la memoria final. Una vez validada, ésta se pasa al tribunal que lo evalúa.

3. Formulación del problema

El principal problema actual de algunos de los proyectos del departamento, es que las pruebas funcionales se llevan a cabo manualmente (mediante plantillas de ficheros de *Excel*).

Precondiciones	Platform	Resultado esperado	Tiempo de subida
		CATEGORÍAS Y ETIQUETAS	
Categorías desplegadas en UNA (Subir Categorias y etiquetas a Alfresco)		Se crean las categorías gestionadas por UNA en AA	
Se crea una categoría en UNA		Se crea la categoría en AA	
Se modifca una categoría en UNA		Se modifca la categoría en AA	
Se edita la traducción de una categoría en UNA		Se modifca la traducción de la categoría en AA	
Se crea una etiqueta con la propiedad "lang" informada en UNA		Se crea una etiqueta con la propiedad "lang" informada en AA	
Se edita el texto de una etiqueta con la propiedad "lang" informada en UNA		Se modifica el texto de la etiqueta en AA; OJO: en Alfresco se crea una nueva con el cambio (para no perder la etiqueta origen)	
Se edita la propiedad "lang" de una etiqueta en UNA añadiendo una locale válido en AA		Se añade/elimina un idioma a la etiqueta en AA; OJO: en Alfresco se crea una nueva con el cambio (para no perder la etiqueta origen)	
		CONTENIDO	

Imagen 1: Ejemplo de plantilla de juego de pruebas

Esto implica la existencia de errores humanos y sus correspondientes correcciones que alejan el estado del proceso de pruebas del óptimo. Además, el hacerlas a mano implica tener que hacer un trabajo repetitivo cuando se quieren llevar a cabo las mismas pruebas en entornos y navegadores distintos. Por lo tanto, el proyecto se basa en crear una herramienta de testing de portales web para obtener resultados y generar informes automáticamente.

Hay que matizar que no siempre conviene automatizar las pruebas funcionales. La automatización es una inversión, y como tal, se debe estudiar su viabilidad. Esto se hace mediante un análisis del producto, del proyecto de desarrollo y de la empresa.

4. Alcance

El alcance del proyecto vendrá definido por la evolución que éste siga de cara a la entrega final. En principio, los pasos a seguir serán los siguientes:

1. crear el portal web *dummy* con Symfony.
2. crear un conjunto de pruebas *dummy*.
3. digitalizar los resultados con Selenium/Mink.
4. integrar los informes obtenidos con TestLink - opcional.
5. documentar el proyecto (en cada paso).

Para garantizar la finalización del proyecto a tiempo, el conjunto de pruebas inicial que se hará sobre el portal web *dummy* será sencillo. De este modo se podrá avanzar el proyecto para comprobar que se pueden digitalizar los resultados. Si esto funciona, se decidirá si destinar el tiempo restante en crear un juego de pruebas más completo o intentar integrar los informes con TestLink.

4.1. Posibles obstáculos

Las dificultades con las que podemos encontrarnos durante el desarrollo del proyecto son, sobretodo, temporales. Esto es así porque se lleva a cabo en unos 4 meses, por lo tanto, la gestión del tiempo es clave en el proyecto. Para ello se deberá planificar el proyecto de forma realista y flexible y definir los juegos de pruebas sencillos para tener un producto final funcional. Una vez finalizado este ciclo, se podrán estudiar y mejorar los juegos de pruebas para acercarlos a los óptimos.

Puesto que la herramienta a desarrollar depende de otros sistemas existentes, hay que tener en cuenta su fiabilidad y compatibilidad entre ellos.

Cabe destacar que el limitado conocimiento inicial sobre el proyecto puede aumentar la curva de aprendizaje de la materia. Para ello es clave la comunicación con los miembros del equipo de la empresa teniendo siempre en cuenta el tiempo empleado y disponible.

5. Objetivo y competencias técnicas

El proyecto se basa en crear una herramienta de testing de portales web para obtener resultados y generar informes automáticamente.

De este modo, se minimizarán los posibles errores humanos y se reducirá el tiempo empleado en pruebas repetitivas permitiendo que las pruebas pasen a ser multi-entorno, multi-idioma, multi-navegador y de mayor calidad. En definitiva, se optimiza el proceso de pruebas disminuyendo su tiempo de ejecución y aumentando su fiabilidad.

El portal por el que navegará la herramienta será un portal *dummy*[3] que tenga las opciones básicas de la capa social: login, menús de navegación, buscador con resultados, eventos de click tipo "like", formularios con adjuntos. Esta herramienta de testing navegará por el portal *dummy* según la configuración especificada en un fichero (properties). Este fichero contendrá los distintos tipos de **Pruebas Funcionales** que se podrán llevar a cabo sobre el portal, de modo que el *tester*[4] solamente tenga que escoger los que aplican o no en la prueba a llevar a cabo.

Las herramientas de testing pueden generar informes en distintos *formatos*[5], y el propósito es ofrecer una alternativa nueva a partir de una librería (API) que permita integrar uno de estos formatos con el software *TestLink*[6].

Por lo tanto, los objetivos tecnológicos del proyecto son:

1. Crear una herramienta de automatización de pruebas funcionales para los distintos entornos del proyecto.
2. Desarrollar el objeto de pruebas: portal web *dummy*.
3. Crear un conjunto de pruebas multi-entorno, multi-idioma, multi-navegador y de calidad.
4. Generar informes automáticamente.

Por otro lado, los objetivos empresariales son:

1. Reducir el tiempo empleado en llevar a cabo las pruebas.
2. Aumentar la fiabilidad de las pruebas, minimizar errores.
3. Evitar el trabajo repetitivo y redundante.
4. Reutilizar juegos de pruebas.

5.1. Competencias técnicas

Al haber adoptado el rol de Analista tanto en la Fase de *Análisis de requisitos* como en la Fase de *Especificación*, se capturan y describen detalladamente las funcionalidades de la herramienta a desarrollar mediante reuniones con el cliente (que en nuestro caso era el *team leader*) a la vez que se especifican claramente los requisitos del proyecto y se define el proceso de automatización de pruebas. Estas funciones concuerdan con la competencia de participar en parte de la especificación de los sistemas de información y de comunicación (**CSI4.1**).

Por otro lado, al haber adoptado el rol de Team Leader tanto en la Fase de *Análisis previo*, como en la Fase de *Gestión de proyectos* como en la Fase de *Finalización*, se analizan los sistemas y *metodologías*[7] actuales del equipo del proyecto y se detectan sus necesidades. Después se

[3] De pruebas, implementado por nosotros
[4] Persona que lleva a cabo los tests
[5] Mail, logs, HTML...
[6] http://testlink.org/
[7] Formas de proceder; conjunto de procedimientos racionales utilizados para alcanzar los objetivos que rigen unas tareas que requieren habilidades o conocimientos específicos.

analizan las oportunidades para satisfacer las necesidades del equipo y se define el proyecto conceptualmente. Esto permite estudiar y evaluar el proyecto y tomar una decisión sobre su realización. Además, se revisará el trabajo realizado para poder destacar desviaciones respecto a la planificación inicial del proyecto y redactar las conclusiones. Todo esto se lleva a cabo con un buen clima interpersonal, sin estrés, cumpliendo con el alcance, dentro de los límites temporales y con los costes definidos. Estas funciones concuerdan con las competencias de concebir, desplegar, organizar y gestionar sistemas y servicios informáticos, en contextos empresariales o institucionales, para mejorar sus procesos de negocio, responsabilizarse y liderar su puesta en marcha, y su mejora continua; y valorar su impacto económico y social (**CSI2.2**), haber participado en el diseño, la implementación y el mantenimiento de los sistemas de información y de comunicación (**CSI4.2**) y evaluar ofertas tecnológicas para el desarrollo de sistemas de información y gestión (**CSI3.3**).

Sin embargo, al haber adoptado el rol de Desarrollador en la Fase de *Diseño, Implementación y pruebas*, se crea un portal *dummy* con Symfony, se crea un conjunto de pruebas *dummy*, se ejecutan pruebas con Selenium/Mink y generan informes con los resultados, se tiene la opción de integrar los resultados con Testlink, se ejecutan los casos de prueba y se documenta el proyecto. Estas funciones concuerdan con la competencia de desarrollar soluciones de negocio mediante la implantación y la integración de hardware y software (**CSI3.4**).

Todos los roles descritos han podido adoptarse al ser capaz de trabajar como miembro de un equipo, ya sea como un miembro más, o realizando tareas de dirección con la finalidad de contribuir a desarrollar proyectos con pragmatismo y sentido de la responsabilidad, asumiendo compromisos teniendo en cuenta los recursos disponibles (**G5**).

El proyecto ha podido llevarse a cabo en su totalidad gracias a haber utilizado de forma apropiada teorías, procedimientos y herramientas en el desarrollo profesional de la ingeniería informática en todos sus ámbitos (especificación, diseño, implementación, despliegue -implantación- y evaluación de productos) de manera que se demuestre la comprensión de los compromisos adoptados en las decisiones de diseño (**CT2**).

5.2. Conocimientos

Los conocimientos gracias a los que se podrá llevar a cabo el proyecto son, especialmente, los de las siguientes asignaturas de la especialidad de SI.

La asignatura de Conceptos de Sistemas de la Información (***CSI***) ha ayudado, junto a la asignatura de Ingeniería de Requisitos (**ER**), a participar en parte de la especificación de los sistemas de información y de comunicación (**CSI4.1**). Estos puntos se han reflejado al adoptar el rol de Analista en las Fases de *Análisis de requisitos* y *Especificación*.

Además, junto con los conocimientos adquiridos en las asignaturas de Diseño de Sistemas de la Información (***DSI***) y de Proyecto de Sistemas de la Información (**PSI**) se ha podido concebir, desplegar, organizar y gestionar sistemas y servicios informáticos, en contextos empresariales o institucionales, para mejorar sus procesos de negocio, responsabilizarse y liderar su puesta en marcha, y su mejora continua; y valorar su impacto económico y social (**CSI2.2**). Estas tres asignaturas (CSI, DSI y PSI) también han permitido haber participado en el diseño, la implementación y el mantenimiento de los sistemas de información y de comunicación (**CSI4.2**). Por otro lado, la asignatura de DSI ha permitido, por sí misma, evaluar ofertas tecnológicas para el desarrollo de sistemas de información y gestión (**CSI3.3**). Estos puntos se han reflejado al adoptar el rol de Team Leader en las Fases de *Análisis previo*, *Gestión de proyectos* y *Finalización* del proyecto.

La asignatura de Negocio Electrónico (***NE***) ha aportado la capacidad de desarrollar soluciones de negocio mediante la implantación y la integración de hardware y software (**CSI3.4**). Esto ha quedado reflejado al adoptar el rol de Desarrollador en la Fase de *Diseño, Implementación y Pruebas*.

5.3. Adecuación

Puesto que las organizaciones que invierten en agilizar sus procesos son las que podrán utilizar sus Sistemas de Información para competir estratégicamente y tras observar que en algunos proyectos del departamento de "Digital Channels (Unidad de Tecnología)" de la empresa "everis an NTT DATA Company" se llevan a cabo los juegos de pruebas de forma manual, se identifica una oportunidad de mejora. Ésta oportunidad se halla en el proceso de desarrollo y mantenimiento de portales de diversa índole (conocimiento, intranets, contenidos, capa social) sobre Liferay, PHP, con Sharepoint u Oracle WebCenter Sites para grandes sistemas corporativos.

Al identificar una oportunidad de mejora en el proceso y con la finalidad de agilizarlo, se introduce una innovación basada en crear una herramienta de testing para obtener resultados y generar informes automáticamente. Para conseguirlo, se analizan los requisitos de la organización y se diseñan soluciones escogiendo, adaptando e integrando las herramientas disponibles más adecuadas. Por lo tanto, se actúa de puente entre las necesidades de gestión y las posibilidades que la tecnología ofrece.

El proyecto se lleva a cabo familiarizándose con los distintos tipos de herramientas tecnológicas para construir parte de un sistema de información. Esta parte es la que hace referencia a la gestión de procesos internos, la gestión del conocimiento y la ayuda a la toma de decisiones.

Todo esto se acomete al comprender los procesos operativos y la gestión que llevan a cabo las personas que forman parte del equipo de la organización, características que la especialidad de Sistemas de la Información intenta inculcar a sus estudiantes al cursar sus asignaturas.

Por todo lo argumentado, considero que el proyecto es del todo adecuado como Trabajo de Fin de Grado de la especialidad de Sistemas de la Información.

6. Metodología y rigor

El primer paso es desarrollar un portal *dummy* con las opciones básicas de la capa social: login, menús de navegación, buscador con resultados, eventos de click tipo "like", formularios con adjuntos. A continuación se estudiará la herramienta escogida para posteriormente desarrollar los tests que se correrán sobre el portal.

Los tests funcionales comprueban la ejecución, revisión y retroalimentación de las funcionalidades diseñadas para el software. Prueban y validan que el software hace lo que debe y, sobre todo, lo que se ha especificado. El objetivo es comprobar que el sistema cumple con los requisitos funcionales: navegación, entrada de datos, procesamiento y obtención de resultados.

Esta metodología trata de enfocar las pruebas funcionales en los requisitos funcionales, basándose en los Casos de Uso y las reglas del negocio. Después pretende verificar la aplicación y sus procesos internos analizando la salida (resultados) mediante la interacción vía interfaz gráfica de usuario (técnica *caja negra*[8]).

Por lo tanto, el procedimiento se basa en:

1. Ejecutar cada Caso de Uso, flujo de caso de uso o función usando datos válidos e inválidos
2. Desarrollar la funcionalidad probada (parte del portal *dummy*).
3. Verificar:
    a. que los resultados esperados ocurren al usar datos válidos.
    b. que los mensajes de error y precaución se despliegan apropiadamente al usar datos inválidos.
    c. que cada regla de negocio se aplica apropiadamente.

Este procedimiento se da por finalizado cuando:

1. Se han ejecutado todas las pruebas planeadas.
2. Se han identificado y considerado todos los defectos.

Como puede observarse, el desarrollo de las funcionalidades del portal se hace a medida que se ejecutan los Casos de Uso. Esto se debe al uso de la metodología *Behavior Driven Development* explicada más adelante.

Para finalizar, se intentará integrar la salida de los juegos de pruebas con el programa *TestLink*. Esto conlleva buscar una librería (API) que nos permita integrar alguna de las salidas de los juegos de pruebas con el gestor de tests.

Las herramientas con las que se hará un seguimiento del estado del proyecto serán:

- Google Drive [GD12]: carpetas compartidas con la documentación
- GitHub [GH08]: sistema de control de versiones del código
- e-mail [GM04]: concertar reuniones y plantear dudas al ponente
- Gantter [Gan09]: programación del proyecto

Las validaciones se harán semanalmente con el tutor del proyecto para comprobar su desarrollo y quincenalmente con el ponente para llevar a cabo su seguimiento con el soporte de las herramientas especificadas. Por lo tanto, la metodología utilizada ha sido una metodología ágil parcial, pues se han llevado a cabo pequeños ciclos en los que se han comprobado y verificado los resultados con el team-leader y también unos ciclos más grandes a modo de punto de control, pero no se han planificado reuniones diarias de 15 minutos ni se ha utilizado un tablón para introducir las tareas y problemas.

[8] Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

6.1. Tests y Comportamiento

6.1.1. TDD: Test-Driven Development

En esta sección se va a explicar el Desarrollo Basado en Testeo (Test-Driven Development: TDD en adelante). El TDD es un método de programación extrema en el que se desarrolla un sistema software en pequeñas iteraciones basadas en tests [Tor12]. Esto se opone al desarrollo de software que permite ampliar el software con partes que no hayan demostrado cumplir los requisitos.
Tal y como define Kent Beck [Bec03], el ciclo de desarrollo del TDD consta de 6 partes principales:

1. Añadir un test.
2. Ejecutar todos los tests y ver si el test añadido falla.
3. Escribir el código.
4. Ejecutar los tests.
5. Refactorizar el código.
6. Repetir.

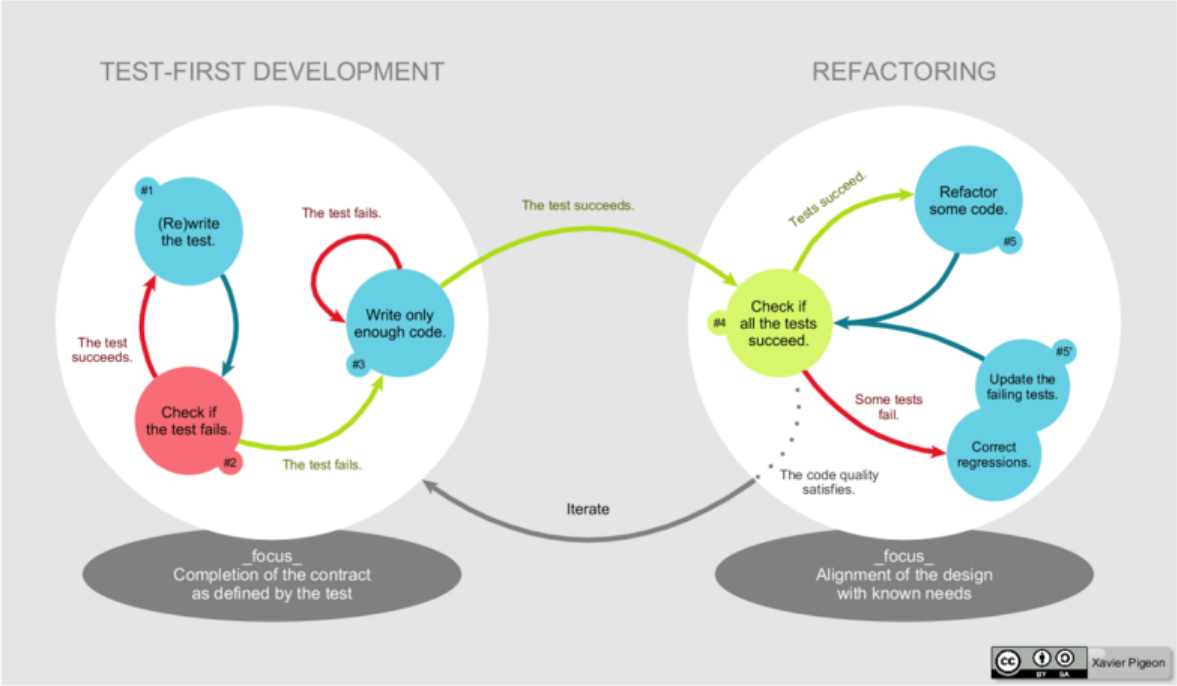


Imagen 3: Ciclo de vida del método Test-Driven Development [Pig15]

6.1.1.1. Añadir un test

Cada funcionalidad comienza escribiendo un test. Este test ha de definir una función (o su mejora) lo más brevemente posible. Para escribir un test, el desarrollador debe entender con claridad la especificación de la funcionalidad y sus requisitos. A través de los casos de uso y las historias de usuario, el desarrollador puede cubrir los requisitos y las condiciones excepcionales. El test puede ser nuevo o la modificación de un test existente. La gran diferencia entre el TDD y el método tradicional (escribir tests unitarios una vez el código haya sido escrito) es que el TDD obliga al desarrollador a centrarse en los requisitos antes de escribir el código.

6.1.1.2. Ejecutar todos los tests y ver si el test añadido falla

Aquí se valida que el entorno del test funcione correctamente, que el nuevo test no pase con éxito sin requerir algo de código nuevo y que la funcionalidad requerida no exista. También se elimina la posibilidad de que el nuevo test siempre sea un éxito y que, por lo tanto, sea inútil. De este modo, el desarrollador sabe que está testeando la restricción correcta y que solo pasa en los casos oportunos.

6.1.1.3. Escribir el código

En este momento se escribe el código que haga que el test sea un éxito. El código escrito no tiene que ser perfecto y puede pasar de forma poco elegante (más adelante se perfeccionará). Prima pasar el test, por lo que el desarrollador no tiene que implementar nada más que la funcionalidad que el test comprueba.

6.1.1.4. Ejecutar los tests

Si todos los tests son un éxito, el desarrollador sabe que el código cumple los requisitos. En caso contrario debe ajustarse el código hasta que los cumpla.

6.1.1.5. Refactorizar el código

El código base crece con cada iteración, por lo que ha limpiarse con regularidad durante el TDD. Se contemplan acciones como:

- Mover el código a un emplazamiento más lógico.
- Eliminar duplicidades.
- Los nombres de los objetos, clases, módulos, variables y métodos han de representar su propósito con claridad.
- Aumentar los cuerpos de los métodos al añadir nuevas funcionalidades.
- Separar métodos y nombrar con lógica para mejorar su legibilidad y mantenibilidad.
- Etc.

6.1.1.6. Repetir

Para incrementar las funcionalidades, debe repetirse el ciclo con un nuevo test. El tamaño de los pasos ha de ser mínimo (entre 1-10 ediciones entre cada ejecución del test). Si el código escrito hace que fallen otros tests o si no se satisface el nuevo test con rapidez, el desarrollador debe deshacer o revertir el código. La integración contínua ayuda a tener puntos de control revertibles. Cuando se usen librerías externas, hay que evitar hacer incrementos tan pequeños que equivalgan a testear la librería en sí (a no ser que se dude de la fiabilidad de la misma).

6.1.2. BDD: Behavior-Driven Development

En esta sección se va a explicar el Desarrollo Basado en Comportamiento (Behavior-Driven Development: BDD en adelante). El concepto de BDD fue introducido por Dan North en 2006 [Nor06] como respuesta a los problemas que surgían al enseñar TDD. Se trata de un proceso de desarrollo de software que evoluciona del TDD. Por lo tanto, como sintetiza Raúl Hernández [Her16], en el BDD también se escriben las pruebas antes de implementar el código, pero en vez de tratarse de

pruebas unitarias, las pruebas del BDD verifican el comportamiento del código desde el punto de vista de negocio. El BDD se centra en los siguientes 5 puntos:

1. Dónde comenzar el proceso.
2. Qué testear y qué no testear.
3. Cuánto testear de una tirada.
4. Cómo llamar a los tests.
5. Cómo entender por qué un test falla.

El BDD matiza que el test de cualquier unidad de software debe especificarse como un "deseo de comportamiento" de dicha unidad. Ese "deseo de comportamiento" consiste en los requisitos específicos del negocio, es decir, el comportamiento que tiene valor de negocio para la entidad que encargue la construcción de la unidad de software. Por lo tanto, siguiendo el principio del TDD, en el BDD lo que se hace es:

1. Escribir pruebas que verifiquen el correcto comportamiento del código desde el punto de vista del negocio.
2. Escribir el código de la funcionalidad que haga que las pruebas pasen con éxito.
3. Refactorizar el código.

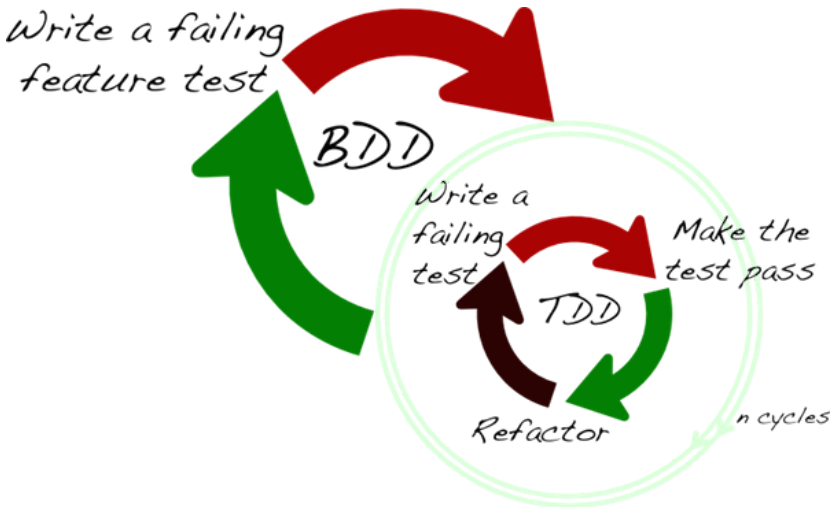


Imagen 4: Ciclo de vida del método Behavior-Driven Development [Env15]

6.1.2.1. Especificación del comportamiento

El modo en que el BDD especifica el deseo de comportamiento es un formato semi-formal heredado de las especificaciones de las historias de usuario. Por ello, el BDD especifica que analistas de negocio y desarrolladores deben trabajar codo con codo en este área y especificar el comportamiento a modo de historias de usuario. Estas historias de usuario deben escribirse explícitamente en un documento concreto. Cada historia de usuario debe, de una forma u otra, seguir la siguiente estructura:

6.1.2.1.1. Título

Es el título de la história de usuario. Debe ser claro y explícito

6.1.2.1.2. Narrativa

Es una sección introductoria corta que especifica:

- Quién (rol de negocio o de proyecto) es el *stakeholder* principal que lleva a cabo la história.
- Qué efecto quiere que tenga la história el *stakeholder*.

- Qué valor de negocio derivará del efecto al *stakeholder*.

6.1.2.1.3. Escenarios o criterios de aceptación

Es una descripción de cada caso específico de la Narrativa. Dicho escenario sigue la siguiente estructura:

1. Especifica la condición inicial que se asume como cierta al comienzo del escenario. Puede tratarse de una o de varias.
2. Establece el evento que acciona el inicio del escenario.
3. Establece la salida esperada. Puede tratarse de una o de varias cláusulas.

Sin embargo, el BDD no tiene unos requisitos formales sobre cómo deben escribirse dichas historias de usuario. Pero por otro lugar, se insta a que los equipos que usen BDD, escriba historias de usuario que incluyan lo mencionado de una forma estándar y simple.

En este proyecto se ha seguido el lenguaje sugerido por la herramienta utilizada (Behat), que sigue la plantilla recomendada por Dan North en 2007 [Nor07].

7. Planificación temporal

En este apartado se describen los distintos ámbitos de control del proyecto: se planean y programan las actividades del proyecto con el fin de que el tiempo total, entre su inicio y su finalización, sea el esperado. Los recursos de cada ámbito se definen con el rol que los llevará a cabo.

7.1. Planificación inicial

7.1.1. Descripción de tareas y recursos

Las fases del proyecto y el rol que las lleva a cabo son las especificadas a continuación. Cabe destacar que, pese a la variedad de roles especificados, todos han sido llevados a cabo por el autor del TFG a excepción del de la Fase I. Se indica entre paréntesis el porcentaje de tiempo dedicado a cada fase respecto al total del proyecto.

Fase I.- Análisis previo (12'6%)

Rol: **Team leader**.
Todo proyecto es un tipo de inversión y, como tal, requiere una justificación. Se analizan los sistemas y *metodologías*[9] actuales del equipo del proyecto y se detectan sus necesidades. Al tratarse de un proyecto definido por el propio *Team Leader* para su equipo, éste se basa en la información existente, el juicio común y la opinión que otorga la experiencia para saber las metodologías de automatización de pruebas, el modo en que se realizan las pruebas y las necesidades específicas de la empresa. Después se analizan las oportunidades para satisfacer las necesidades del equipo y se define el proyecto conceptualmente. Esto se lleva a cabo para poder estudiar y evaluar el proyecto y así tomar una decisión sobre su realización. Un estudio muy profundo implica un mayor gasto de recursos que, en caso de no llevarse a cabo, se darían por perdidos. De aquí la importancia de definir bien su alcance.

Fase II.- Gestión de proyectos (20'1%)

Rol: **Team leader**.
Para que un proyecto pueda ser desarrollado con un buen clima interpersonal, sin estrés, cumpliendo con el alcance, dentro de los límites temporales y con los costes definidos, la persona a cargo ha de estar cualificada. Esto significa que ha de tener la capacidad de planificar, captar, dinamizar, organizar talentos y administrar recursos.

Con esta finalidad, el proyecto ha sido gestionado para constar de las siguientes partes, divididas en las distintas entregas de la asignatura de Gestión de Proyectos (GEP):

1. Definición del alcance.
2. Planificación temporal.
3. Gestión Económica y sostenibilidad.
4. Presentación preliminar.
5. Pliego de condiciones de la especialidad de Sistemas de la Información.

Llevadas a cabo bajo la supervisión del Ponente mediante:

[9] Formas de proceder; conjunto de procedimientos racionales utilizados para alcanzar los objetivos que rigen unas tareas que requieren habilidades o conocimientos específicos.

6. Seguimiento de la planificación y ejecución del proyecto (de principio a fin).
7. Análisis de desviaciones y cumplimiento de la planificación y uso de recursos.

Fase III.- Análisis de Requisitos (1'3%)

Rol: **Analista**.
Se capturan y describen detalladamente las funcionalidades de la herramienta. Se obtienen mediante reuniones con el cliente, que en nuestro caso será el *team leader*.

Fase IV.- Especificación (2'5%)

Rol: **Analista**.
Se especifican claramente los requisitos del proyecto y se define el proceso de automatización de pruebas. En función de la fase en la que se encuentra el proyecto, las partes son:

- Especificación de Requisitos: acuerdo entre usuario y desarrollador del sistema
- Especificación de Diseño: acuerdo entre diseñador e implementador
- Especificación de Módulos: acuerdo entre programadores usuarios del módulo y programadores implementadores del módulo.

Fase V.- Diseño, Implementación y Pruebas (60'4%)

Rol: **Desarrollador**.
Se desarrolla la herramienta del proyecto en 6 partes bien diferenciadas:

1. Crear el portal *dummy* con Symfony.
2. Crear un conjunto de pruebas *dummy.*
3. Ejecución de pruebas con Selenium/Mink y generación de informes con los resultados.
4. Integrar los informes obtenidos con TestLink - opcional.
5. Documentar el proyecto (en todos los pasos anteriores).

Fase VI.- Finalización (3'1%)

Rol: **Team leader**.
Se revisa el trabajo realizado para poder destacar las desviaciones respecto a la planificación inicial del proyecto y redactar las conclusiones.

7.1.2. Análisis temporal

Al tratarse de un Trabajo de Fin de Grado (TFG) el marco temporal queda limitado al cuatrimestre universitario 2015/16-Q2, esto es desde 15 febrero - 27 junio 2016.

7.1.2.1. Descripción diagrama de Gantt:

En el periodo inicial [*18 Enero - 20 Febrero 2016*], las actividades son llevadas a cabo por el *Team Leader* del equipo para proponer el proyecto como TFG.

En el segundo periodo [*18 Febrero - 01 Abril 2016*], las actividades las lleva a cabo el autor del TFG en el rol del *Team Leader* desde el inicio de la asignatura de GEP hasta su finalización.

En el tercer periodo [*07 Marzo - 15 Marzo 2016*], las actividades las lleva a cabo el autor del TFG en el rol de Analista.

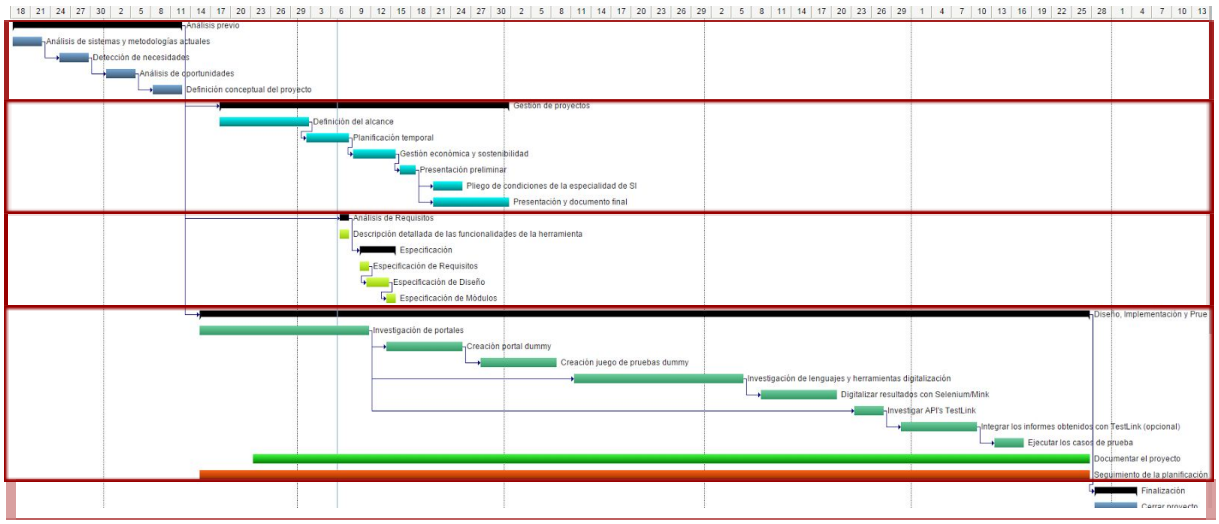
En el cuarto periodo [*15 Febrero - 27 Junio 2016*], las actividades las lleva a cabo el autor del TFG en el rol de Desarrollador. Este periodo se inicia en el momento de la incorporación a la empresa y finaliza con la presentación del TFG frente al tribunal.

Debido a la poca formación inicial en el tema a desarrollar, se ha reservado un tiempo considerable para la investigación de cada uno de los apartados (Investigación de portales e Investigación de lenguajes y herramientas). Se ha reservado un tiempo prudencial al final del proyecto para las posibles desviaciones que pueda sufrir la organización inicial.

La documentación del proyecto se lleva a cabo desde prácticamente su inicio y el seguimiento de la planificación y ejecución del proyecto se lleva a cabo desde el primer día.

El periodo final [*28 Junio - 04 Julio 2016*] es llevado a cabo por el autor del TFG en el rol de *Team Leader*.

7.1.2.2. Diagrama de Gantt:



	Nombre	Duración	Inicio	Fin	Predecesoras	Recursos
1	**Análisis previo**	20d?	**18/01/2016**	**12/02/2016**		**Team Leader**
2	Análisis de sistemas y metodologías actuales	5d?	18/01/2016	22/01/2016		Team Leader
3	Detección de necesidades	5d?	25/01/2016	29/01/2016	2	Team Leader
4	Análisis de oportunidades	5d?	01/02/2016	05/02/2016	3	Team Leader
5	Definición conceptual del proyecto	5d?	08/02/2016	12/02/2016	4	Team Leader
6	**Gestión de proyectos**	32d?	**18/02/2016**	**01/04/2016**	**1**	**Team Leader**
7	Definición del alcance	10d?	18/02/2016	02/03/2016		Team Leader
8	Planificación temporal	5d?	02/03/2016	08/03/2016	7	Team Leader
9	Gestión económica y sostenibilidad	5d?	09/03/2016	15/03/2016	8	Team Leader
10	Presentación preliminar	3d?	16/03/2016	18/03/2016	9	Team Leader
11	Pliego de condiciones de la especialidad de SI	5d?	21/03/2016	25/03/2016	10	Team Leader
12	Presentación y documento final	10d?	21/03/2016	01/04/2016	10	Team Leader
13	**Análisis de Requisitos**	2d?	**07/03/2016**	**08/03/2016**	**1**	**Analista**
14	Descripción detallada de las funcionalidades de la herramienta	2d?	07/03/2016	08/03/2016		Analista
15	**Especificación**	4d?	**10/03/2016**	**15/03/2016**	**13**	**Analista**
16	Especificación de Requisitos	2d?	10/03/2016	11/03/2016		Analista
17	Especificación de Diseño	2d?	11/03/2016	14/03/2016	16	Analista
18	Especificación de Módulos	2d?	14/03/2016	15/03/2016	17	Analista
19	**Diseño, Implementación y Pruebas**	96d?	**15/02/2016**	**27/06/2016**	**1**	**Desarrollador**
20	Investigación de portales	20d?	15/02/2016	11/03/2016		Desarrollador
21	Creación portal dummy	10d?	14/03/2016	25/03/2016	20	Desarrollador
22	Creación juego de pruebas dummy	10d?	28/03/2016	08/04/2016	21	Desarrollador
23	Investigación de lenguajes y herramientas digitalización	20d?	11/04/2016	06/05/2016	20	Desarrollador
24	Digitalizar resultados con Selenium/Mink	10d?	09/05/2016	20/05/2016	23	Desarrollador
25	Investigar API's TestLink	5d?	23/05/2016	27/05/2016	20	Desarrollador
26	Integrar los informes obtenidos con TestLink (opcional)	10d?	30/05/2016	10/06/2016	25	Desarrollador
27	Ejecutar los casos de prueba	5d?	13/06/2016	17/06/2016	26	Desarrollador
28	Documentar el proyecto	90d?	23/02/2016	27/06/2016		Desarrollador
29	**Finalización**	5d?	**28/06/2016**	**04/07/2016**	**19**	**Team Leader**

Imagen 5: Diagrama de Gantt de la planificación inicial (Anexo I)

7.1.3. Recursos

A continuación se detallan las funciones de cada uno de los recursos humanos del proyecto:

- Team Leader (35'8%): Encargado de planificar, monitorizar y controlar las tareas.

- Analista (3'8%): Encargado de llevar a cabo el análisis de requisitos y la especificación.
- Desarrollador (60'4%): Encargado de diseñar, implementar y testear la herramienta.
- Ponente (100%): Encargado del seguimiento de la planificación y de la ejecución del proyecto.

Al tratarse de un TFG, se ha de tener en cuenta el rol del director del proyecto, cuya participación es de alrededor de un 20% adicional de cada una de las actividades.

7.1.4. Secuencia lógica y dependencias

Como se puede observar en el la *Figura 1*, las actividades del segundo, tercer y cuarto periodos (*Gestión de proyectos*, *Análisis de requisitos* y *Diseño, Implementación y Pruebas*) dependen de la finalización de las actividades del primero (*Análisis previo*). Esto es así porque inicialmente lo imprescindible es ver la viabilidad del proyecto. Una vez aceptado éste, las actividades de los tres periodos mencionados pueden llevarse a cabo en paralelo. Esta paralelización de procesos se debe a que la fase de *Diseño, implementación y Pruebas* tiene una inversión temporal inicial muy alta en la *Investigación de portales*.

Dentro de cada una de las fases, las tareas suelen llevarse a cabo en *cascada*[10] a excepción de las siguientes:

- *Pliego de condiciones de la especialidad de SI* + *Presentación y documento final*, que pueden realizarse en paralelo una vez finalizada la *presentación preliminar* de la fase de *Gestión de proyectos*.
- *Creación del portal dummy* + *Investigación de lenguajes y herramientas de digitalización* + *Investigación de API's de TestLink*, que pueden realizarse en paralelo una vez finalizada la *Investigación de portales*.
- *Documentación del proyecto* + *Seguimiento de la planificación y ejecución del proyecto* que se llevan a cabo en paralelo con toda la fase de *Diseño, Implementación y Pruebas*.

7.2. Valoración de alternativas y plan de acción

En la planificación inicial se tomaron dos precauciones para garantizar la finalización del proyecto a tiempo:

- Alargar los periodos de investigación para compensar la falta de experiencia en el tema y la tecnología con la que se iba a trabajar.
- Acotar el TFG una semana menos para dejar margen suficiente al final por si se sufría alguna desviación.

7.2.1. Análisis de alternativas para la resolución de problemas

A continuación se enumeran las posibles causas de alguna desviación, su solución y el efecto en la duración total del proyecto:

	Causa	Solución	Gravedad[11]

[10] [Men08] Al final de cada fase se revisan las tareas de trabajo y productos; Para poder pasar a la siguiente fase se tienen que haber conseguido todos los objetivos de la fase anterior; No hay apenas comunicación entre las fases.
[11] Valores de gravedad: Alta, Media y Baja.

1	Poca experiencia con la tecnología, las herramientas y el lenguaje de programación empleados	Consultar internet y libros para empezar por la implementación de las funcionalidades más sencillas	Media
2	Mala planificación inicial	Detectar las desviaciones lo antes posible y actuar en consecuencia	Media
3	Aparición de nuevas tareas	Priorizar y paralelizar tareas	Media
4	Fracaso en la integración de resultados con TestLink	Buscar herramienta alternativa	Baja

Tabla 2: Causas, Soluciones y Efectos en la duración total del proyecto a desviaciones eventuales.

En el caso de surgir alguna desviación respecto a la planificación inicial del proyecto, se invertiría más tiempo para poder solucionar el desvío con celeridad y disminuir la gravedad. Esto equivale a invertir más recursos humanos (horas) en la realización y desarrollo del proyecto. Durante los periodos desviados, se priorizará la solución de los problemas con mayor repercusión en la planificación inicial del proyecto reduciendo o incluso prescindiendo de tareas secundarias.

7.3. Seguimiento

7.3.1. Cambios producidos respecto a la planificación inicial

La semana del 7 de abril, se tuvo el primer retraso (de una semana) en el proyecto.

- Ese día, se debería tener ya el portal *dummy* de Symfony y estar acabando la definición de los juegos de prueba. La realidad era que se seguía con problemas al desarrollar el portal:
  - obligatoriedad de entregas semanales de la asignatura de GEP del TFG (Gestión de Proyectos)
  - problemas con las versiones de Symfony: se empezó con la última versión (3.0), pero la guía del libro que se seguía recomendaba instalar una serie de vendors que eran incompatibles con la última versión, así que se tuvo que bajar a la 2.8 y empezar de nuevo. Ya que se empezaba de nuevo, se decidió realizar un proyecto "nuevo" sin las pruebas que se habían ido haciendo para aprender.
  - El nuevo proyecto se decidió llevarlo a cabo en el portátil personal del autor del TFG en vez de en el ofrecido por la empresa, puesto que casi siempre lo lleva encima (poco peso), es más potente y facilita todas las instalaciones (OSX) que en Windows son más tediosas. El día 6 de abril se dedicó a intentar conectar el ordenador a la red interna de Everis, puesto que por wifi era inviable descargarse los programas e instalarlos en un mismo día. Se tuvo que comprar un adaptador, bajar e instalar unos drivers nuevos e instalar las VPN's interna y externa de la empresa (posible gracias a unos paquetes que se facilitaron).
  - Una vez conseguido conectarlo a la red, se bajó:
    - MAMP: gestionar el servidor y alojar el proyecto.
    - Navicat: Gestionar la BD.
  - Puesto que la BD ya estaba creada desde el explorador con phpmyadmin, se decidió exportarla e integrarla en el nuevo proyecto. Esto enseñó a generar entidades en una BD existente. Para poder integrar el proyecto con la BD exportada, se necesitó Doctrine.

La semana del 13 de abril se profundiza en los formularios de Symfony y la persistencia con Doctrine, pero la curva de aprendizaje alarga mucho su implementación, por lo que hasta pasado un mes no se podría tener finalizado.
Por ello, por la corta duración del proyecto y por el nivel de conocimiento adquirido hasta la fecha en Symfony, se decide que, el lunes 18 Abril, el equipo de trabajo facilitará un proyecto sencillo con las funcionalidades básicas implementadas en Symfony. De este modo se puede desbloquear la creación del portal y centrarse en seguir adelante con el proyecto.
Así pues, se focaliza el esfuerzo en el estudio y desarrollo de los Juegos de Pruebas.

La semana del 22 de abril se decide finalizar un ciclo sencillo de todo el proceso.

- Se ha aprendido el funcionamiento de Behat y Mink para poder llevar a cabo la prueba. En cuanto se ha finalizado una prueba con éxito, se ha pasado al siguiente paso sin profundizar más.
- Se ha automatizado la prueba:
    - creando un form en twig para la salida de la prueba
    - creando una nueva ruta a través de la que se procede (automáticamente) a la ejecución de la prueba

El problema que se encuentra, es que al intentar verter el resultado de toda la prueba en una variable, al devolver esta variable solamente aparece parte de la misma. Además, el resultado en la variable está mal parseado.

Se ha explorado otro modo de devolver la prueba para obtenerla toda en la misma página y correctamente parseada:

- Se guarda la salida en un fichero (llamado según la fecha y hora de ejecución) que se ordena automáticamente en la carpeta "tests_output/nombreDelTestEjecutado"
- Se muestra el fichero en la página

El problema es que, al tardar mucho en recibir la segunda parte del fichero (comprobación de los tests), el thread pierde el hilo y no rellena el fichero del todo.
Se propone estudiar la programación multihilo en PHP.

La semana del 6 de mayo se intentó finalizar un ciclo sencillo de todo el proceso:

- La programación multihilo de PHP resultó ser demasiado profunda para el tiempo del que se dispone, por lo que se exploraron otras vías para solucionar el problema:
    - Pipes: leyendo todo lo que entraba y salía del comando PHP que ejecutaba los tests. – seguíamos viendo la primera mitad del fichero a través de las pipes
    - Aumentar el tiempo de espera de la página web para que no diese un timeoutException(). – no cambió nada
    - Bloquear el fichero abierto para evitar que sea leído antes de finalizar los tests. – no cambió nada
- Puesto que únicamente pasaba al ejecutarlo desde la web, en este punto se planteó que podría ser un tema de control de las capas.
    - La solución propuesta fue dar el control de la ejecución del comando al ordenador:
        - la página web carga una entrada en el crontab para que se ejecute al siguiente minuto
        - una vez ejecutado, lo elimina del crontab. – lo malo de la solución es que, en el peor de los casos, hay que esperar un minuto entero hasta la ejecución del comando, y a este tiemo hay que añadir el de ejecución de las pruebas. En otras palabras, no sería instantáneo.
- Otra hipótesis fue que hasta que el controlador no devolvía el control a la vista, el fichero estaba bloqueado y, por lo tanto, lo que se devolvía a la vista desde el controlador era solo la primera mitad.

- La solución propuesta fue hacer dos llamadas desde la vista al controlador:
  - la primera para ejecutar el comando y
  - la segunda para abrir y leer el fichero. – de este modo, el fichero se habrá escrito por completo cuando se lea su contenido. Esto se podría hacer cuando el equipo compartiese el proyecto (se espera desde el 18 de abril, aunque se entregó la primera versión con una página de login el lunes lunes 9 de mayo).

Esa semana se destinó a hacer cursos por internet de jQuery (con pinceladas de AJAX) a través de:

- codeschool (try jQuery, jQuery: The ReturnFlight) y
- codecademy (jQuery)

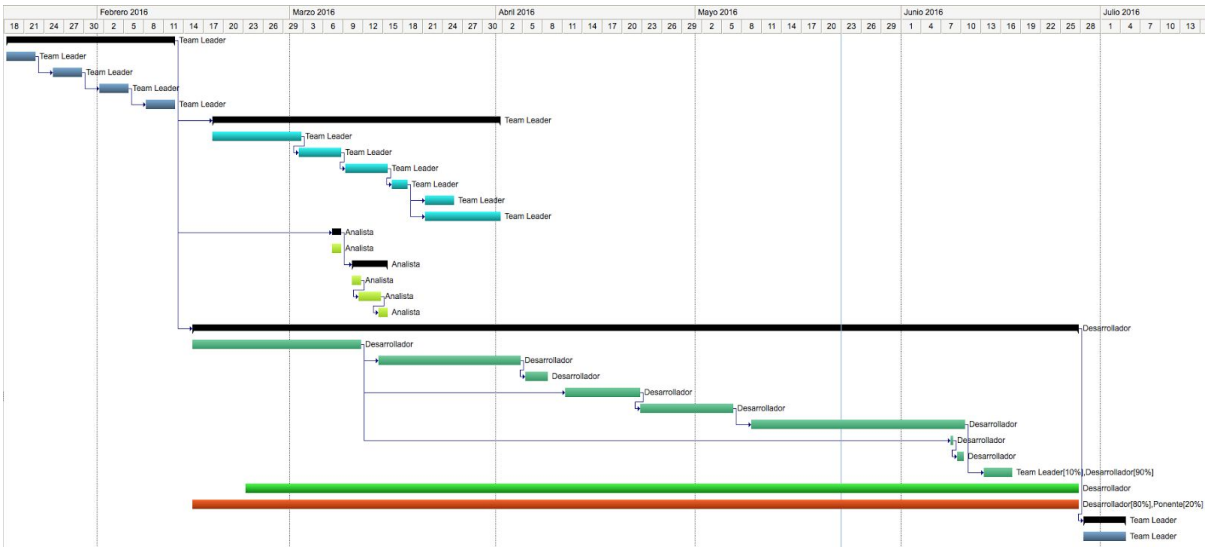
Los motivos para hacer estos cursos fueron:

- no quedarse estancado esperando el proyecto
- el proyecto que se proporcionaría hace uso de jQuery y AJAX

Las siguientes semanas (hasta el 20 de mayo), se destinan a hacer más ciclos de todo el proceso intentando finalizar los juegos de prueba.

7.3.2. Planificación definitiva

Pese a los retrasos sufridos en lo referente al portal *dummy*, el haber paralelizado actividades ha permitido garantizar la fecha de finalización del mismo con éxito, con la única falta de la integración opcional con testLink. Así pues, la planificación definitiva ha sido la siguiente:



		Nombre	Duración	Inicio	Fin	Predecesoras	Recursos
1		**Análisis previo**	20d?	**18/01/2016**	**12/02/2016**		**Team Leader**
6		**Gestión de proyectos**	32d?	**18/02/2016**	**01/04/2016**	**1**	**Team Leader**
7		Definición del alcance	10d?	18/02/2016	02/03/2016		Team Leader
8		Planificación temporal	5d?	02/03/2016	08/03/2016	7	Team Leader
9		Gestión económica y sostenibilidad	5d?	09/03/2016	15/03/2016	8	Team Leader
10		Presentación preliminar	3d?	16/03/2016	18/03/2016	9	Team Leader
11		Pliego de condiciones de la especialidad de SI	5d?	21/03/2016	25/03/2016	10	Team Leader
12		Presentación y documento final	10d?	21/03/2016	01/04/2016	10	Team Leader
13		**Análisis de Requisitos**	2d?	**07/03/2016**	**08/03/2016**	**1**	**Analista**
14		Descripción detallada de las funcionalidades de la herramienta	2d?	07/03/2016	08/03/2016		Analista
15		**Especificación**	4d?	**10/03/2016**	**15/03/2016**	**13**	**Analista**
16		Especificación de Requisitos	2d?	10/03/2016	11/03/2016		Analista
17		Especificación de Diseño	2d?	11/03/2016	14/03/2016	16	Analista
18		Especificación de Módulos	2d?	14/03/2016	15/03/2016	17	Analista
19		**Diseño, Implementación y Pruebas**	96d?	**15/02/2016**	**27/06/2016**	**1**	**Desarrollador**
20		Investigación de portales	20d?	15/02/2016	11/03/2016		Desarrollador
21		Creación portal dummy	16d?	14/03/2016	04/04/2016	20	Desarrollador
22		Creación juego de pruebas dummy	4d?	05/04/2016	08/04/2016	21	Desarrollador
23		Investigación de lenguajes y herramientas digitalización	9.88d?	11/04/2016	22/04/2016	20	Desarrollador
24		Digitalizar resultados con Selenium/Mink (1er ciclo)	10.13d?	22/04/2016	06/05/2016	23	Desarrollador
25		Digitalizar resultados con Selenium/Mink (resto ciclos)	24.5d?	09/05/2016	10/06/2016	24	Desarrollador
26		Investigar API's TestLink	1d?	08/06/2016	08/06/2016	20	Desarrollador
27		Integrar los informes obtenidos con TestLink (opcional)	1.13d?	09/06/2016	10/06/2016	26	Desarrollador
28		Crear Demo	4.88d?	13/06/2016	17/06/2016	25	Team Leader[10%],Desarrollador[90%]
29		Documentar el proyecto	90d?	23/02/2016	27/06/2016		Desarrollador
30		Seguimiento de la planificación y ejecución del proyecto	96d?	15/02/2016	27/06/2016		Desarrollador[80%],Ponente[20%]
31		**Finalización**	5d?	**28/06/2016**	**04/07/2016**	**19**	**Team Leader**
32		Cerrar proyecto	5d?	28/06/2016	04/07/2016		Team Leader

Imagen 6: Diagrama de Gantt de la planificación definitiva

7.3.3. Efectos del cambio

7.3.3.1. Efectos sobre los objetivos o desarrollo del proyecto

Los objetivos tecnológicos del proyecto se han cumplido satisfactoriamente, ya que se trataba de:

1. Crear una herramienta de automatización de pruebas funcionales para los distintos entornos del proyecto.
2. Desarrollar el objeto de pruebas: portal web *dummy*.
3. Crear un conjunto de pruebas multi-entorno, multi-idioma, multi-navegador y de calidad.
4. Generar informes automáticamente.

Los objetivos empresariales del proyecto también se han alcanzado con éxito, ya que constaban de:

5. Reducir el tiempo empleado en llevar a cabo las pruebas.
6. Aumentar la fiabilidad de las pruebas, minimizar errores.
7. Evitar el trabajo repetitivo y redundante.
8. Reutilizar juegos de pruebas.

Sin embargo, hay que mencionar que no se han podido alcanzar los objetivos tecnológicos y empresariales con TestLink. El motivo principal es la versión de Behat utilizada (3.0), que ya no incorpora la funcionalidad de escoger el formato de salida de las pruebas funcionales. Por lo tanto queda como trabajo futuro la integración con TestLink.

7.3.3.2. Efectos sobre los costes del proyecto

El único coste no valorado inicialmente fue el del adaptador de red del portátil personal, pero que al ser para el uso y disfrute del autor del TFG (incluso tras la finalización del TFG), no se considera un coste del proyecto.

7.3.3.3. Punto actual del proyecto sobre la planificación

Pese a los retrasos sufridos en lo referente al portal *dummy*, el haber paralelizado actividades ha permitido garantizar la fecha de finalización del mismo con éxito.

7.3.4. Metodología y rigor

La metodología y rigor iniciales se han seguido en el orden estipulado, con la diferencia de que en vez de desarrollar el portal *dummy* de 0, para no desviar en exceso la planificación inicial, se decidió ceder el esqueleto de un portal sobre el que poder trabajar.

Así pues, el primer paso fue adaptar el portal *dummy* con las opciones básicas de la capa social: login, menús de navegación, eventos de click tipo "like", formularios con adjuntos. A continuación se estudió la herramienta escogida para posteriormente desarrollar los tests que corren sobre el portal.

Los tests funcionales comprueban la ejecución, revisión y retroalimentación de las funcionalidades diseñadas para el software. Prueban y validan que el software hace lo que debe y, sobre todo, lo que se ha especificado. Su objetivo es comprobar que el sistema cumple con los requisitos funcionales: navegación, entrada de datos, procesamiento y obtención de resultados.

Esta metodología trata de enfocar las pruebas funcionales en los requisitos funcionales, basándose en los Casos de Uso y las reglas del negocio. Después pretende verificar la aplicación y sus procesos internos analizando la salida (resultados) mediante la interacción vía interfaz gráfica de usuario (técnica *caja negra*[12]).

Por lo tanto, el procedimiento se basa en:

1. Ejecutar cada Caso de Uso, flujo de caso de uso o función usando datos válidos e inválidos
2. Verificar:
    a. que los resultados esperados ocurren al usar datos válidos.
    b. que los mensajes de error y precaución se despliegan apropiadamente al usar datos inválidos.
    c. que cada regla de negocio se aplica apropiadamente.

Este procedimiento se da por finalizado cuando:

1. Se han ejecutado todas las pruebas planeadas.
2. Se han identificado y considerado todos los defectos.

Las herramientas con las que se ha hecho un seguimiento del estado del proyecto han sido:

- Google Drive: carpetas compartidas con la documentación
- GitHub: sistema de control de versiones del código
- e-mail: concertar reuniones y plantear dudas al ponente
- Gantter: programación del proyecto

[12] Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta.

A estas se han añadido otras como:

- XAMPP y MAMPP: para arrancar el servidor apache y la Base de Datos (BBDD en adelante) tanto en windows como en OSX con un entorno más amigable que el terminal.
- Tower: para mantener el control de versiones durante el desarrollo.
- Navicat y Sequel: para manipular la BBDD.
- PhpStorm: para desarrollar.

Las validaciones se han hecho semanalmente con el tutor del proyecto para comprobar su desarrollo y quincenalmente con el ponente para llevar a cabo su seguimiento con el soporte de las herramientas especificadas.

7.3.4.1. Cambios de la metodología propuesta

La metodología estudiada inicialmente y propuesta para desarrollar el proyecto fue lo suficientemente buena para no precisar ningún cambio ni variación. Únicamente se amplió el número de herramientas utilizadas, que fueron descubiertas bajo recomendación a lo largo del proceso.

7.3.5. Justificación de la opción adoptada

Desde el inicio del proyecto, se han paralelizado procesos para evitar quedarse estancado en fases posteriores. Al haber hecho la planificación inicial como si el desarrollo fuese en cascada, no se tuvieron en cuenta los distintos ciclos que se harían para implementar el proyecto. Los ciclos se hacen tanto para facilitar el desarrollo (empezando por requisitos muy sencillos) como para garantizar que se tiene una aplicación finalizada, aunque sea sencilla.
Esta premisa y la "Baja" gravedad de no implementar la integración con TestLink son las que se tuvieron en cuenta al decidir prescindir de ésta integración de resultados con TestLink. Se decidió hacerlo puesto que se trataba de una funcionalidad que se definió siempre como opcional, y la falta de tiempo la mantuvo siempre al margen como posible implementación. El estado actual del proyecto lleva a concluir que finalmente se prescindirá de ella.

8. Gestión económica

8.1. Identificación y estimación de los costes

Estimar es calcular el coste de hacer algo cuando ya se tiene un plan de trabajo. En este apartado se estiman los costes a partir de la duración por unidad de tiempo de las actividades del diagrama de Gantt. Para ello se tienen en cuenta los recursos utilizados.

8.1.1. Costes Directos (CD)

Los CD de un proyecto son los que engloban las actividades del diagrama de Gantt, es decir, los directamente imputables a su ejecución.

8.1.1.1. Recursos Humanos

Puesto que los roles del proyecto se llevan a cabo teniendo en cuenta el número de horas dedicadas por cada uno, el modo de confeccionar el presupuesto será por unidades de recursos. Así pues, en el plan de trabajo se calculan las horas que dedica cada rol al proyecto. Para ello, se calculan los días de trabajo de cada uno estableciendo un porcentaje de dedicación cuando hay tareas compartidas en una misma etapa.

Rol	Team Leader	Analista	Desarrollador	Ponente
Días	20+32+5 = 57	2+4 = 6	96*0,8 = 76,8	96*0,2 = 19,2

Tabla 3: Días dedicados al proyecto por cada rol

Suponiendo 217 días trabajados (año no bisiesto) y 8h diarias, las horas trabajadas anualmente son 1736, Por lo tanto, los costes nos quedan desglosados de la siguiente forma:

Rol	Horas	Salario anual[13] (€)	€/hora	Coste (€)
Team Leader	456	40.000	23	10.506,9
Analista	48	32.000	18,4	884,8
Desarrollador	614,4	18.000	10,4	6.370,5
Ponente	153,6	35.000	20,2	3.096,8
Total Recursos Humanos	1.272			**20.859 €**

Tabla 4: Desglose de costes de cada rol del proyecto

El total de horas corresponde a las horas trabajadas por el Team Leader en el Análisis Previo más las horas del desarrollo del TFG bajo el convenio de cooperación educativa modalidad B.

[13] Aproximación según [PP16]

8.1.1.2. Software

Para poder desarrollar la herramienta y redactar la memoria a distancia, se han usado (siempre que haya sido posible) herramientas online. Teniendo en cuenta que el software se amortiza en unos 3 años, a continuación se desglosan el software empleado en el desarrollo de la herramienta, que al tratarse de software libre o gratuito, su coste y su amortización se reducen a cero:

Producto	**Coste (€)**	**Amortización proyecto (€) - 5 meses -**
Cite This For Me **Firefox** **Gantter** **Google Chrome+Drive+Gmail** **Selenium** **Symfony** **TestLink** **Ubuntu 14.04.4 LTS** **VirtualBox** **Xampp**	0	0
PhpStorm **Windows 7**	0 - versión estudiante	0
Trello	0 - versión gratuita	0
Total Software		**0 €**

Tabla 5: Desglose de costes de software

8.1.1.3. Hardware

Puesto que este tipo de proyectos requieren inversión de horas fuera del horario de oficina, se tienen varios lugares de trabajo. En función del emplazamiento, se ha usado uno u otro hardware. Teniendo en cuenta que el hardware se amortiza en unos 4 años, a continuación se desglosa el hardware empleado en el desarrollo del proyecto, su coste y su amortización:

Producto	**Coste (€)**	**Vida útil (años)**	**Amortización proyecto (€) - 5 meses -**
DELL Latitude E5550 i5	1.129	4	117,6
Pantalla AOC E2270SWN	127,59	4	13,3
iMac 2009 27'	1.799	4	0 - amortizado
Total Hardware			**130,9 €**

Tabla 6: Desglose de costes de hardware

Por lo tanto, el total de CD es la suma de los recursos humanos, del software y del hardware. Esto da una suma de **20.989,9€**.

8.1.2. Costes Indirectos (CI)

Se engloban los costes que no son directamente atribuibles a una actividad del proyecto.

8.1.2.1. Gastos Generales

Para poder llevar a cabo un proyecto de este tipo, existen una serie de gastos fijos ineludibles que no se consideran en los apartados anteriores:

Producto/Servicio	Coste (€/mes)	Periodo (meses)	Amortización proyecto (€)
Consumo eléctrico	38,16[14]	5	117,6
Acceso internet	26,9	5	134,5
Transporte	20	5	100
Matrícula GEP+TFG+Prac.Ext	252,1	5	1.260,5
Total Gastos Generales			**1.491,4 €**

Tabla 7: Desglose de gastos generales

Por lo tanto, el total de CI es el coste de los gastos generales. Esto da una suma de **1.491,4€**.

8.1.3. Contingencia

Se aplica un porcentaje incremental a los Costes (Directos + Indirectos) para tener en cuenta ciertas pérdidas. Un margen del 10% permite atenuar errores de información incompleta, falta de previsión, formación limitada, descuidos, desviaciones, situaciones incontroladas, etc. Esto desemboca en una contingencia de (20.989 + 1.1491,4)*0,1 = 22.481,3*0,1 = **2.248,1€**, por lo que el total de CD+CI+Contingencia pasa a ser de **24.729,4€**.

8.1.4. Imprevistos

Puesto que el uso de la totalidad del software empleado es online, el retraso en alguna de las actividades planificadas a causa de un imprevisto es nulo. Sin embargo, se tienen en cuenta los costes de los mismos. Los mayores imprevistos a tener en cuenta son el fallo de alguno de los aparatos hardware, puesto que habría que repararlos para continuar el proyecto con normalidad, y la baja por enfermedad puesto que habría una desviación en el proyecto.

	Imprevisto	Reparación (€)	Riesgo (%)	Coste imprevisto (€)
Fallo	**DELL Latitude E5550 i5**	150	10	15
	Pantalla AOC E2270SWN	50	2	1
	iMac 2009 27'	170	40[15]	68
Baja	**Enfermedad**	144[16]	30	43,2

[14] [Esi16]
[15] Ordenador ya amortizado con más de 6 años desde su compra
[16] Media del sueldo diario de todos los roles

Total Imprevistos		**127,2 €**

Tabla 8: Desglose de gastos imprevistos

8.1.5. Coste total del proyecto

Los costes de las distintas partes del proyecto están basados en el Precio de Venta al Público de sus elementos. Esto significa que todos los costes reflejados incluyen el IVA.

Tipo de Coste	Coste Directo	Coste Indirecto	Contingencia	Imprevistos	Total (con IVA)
Coste (€)	20.989,9	1.491,4	2.248,1	127,2	**24.856,6 €**

Tabla 9: Desglose de gastos totales del proyecto

8.2. Control de gestión

Para controlar las posibles desviaciones de presupuesto, se usa el software Trello para indicar las tareas a realizar semanalmente. Paralelamente, se creará una tabla como la que sigue, que se irá rellenando a diario con las horas invertidas por cada uno de los roles:

Rol	13/03/2016
Team Leader	7
Analista	2
Desarrollador	-
Ponente	-

Tabla 10: Desglose de horas de dedicación diaria de cada rol

De este modo, en cada reunión de control quincenal con el Director del proyecto, se puede comprobar la existencia de desviaciones en la organización que pueda llevar a una desviación del presupuesto. Esto se podrá corroborar al final del proyecto creando una tabla como la que sigue, con las horas totales estimadas y dedicadas por cada rol:

Rol	Horas estimadas	Coste Estimado	Horas Reales	Coste Real
Team Leader	456	10.506,9	460	10.599

Tabla 11: Desglose de horas de dedicación diaria de cada rol

Con este método se pueden calcular desviaciones de coste y temporales además de identificar el lugar en el que se producen y el rol para poder corregirlas a tiempo.

9. Estado del arte

9.1. Tipos de tests

Según Javier Gutiérrez [Gut], el objetivo de los procesos de pruebas es repasar las ideas principales sobre las pruebas del software y, en concreto, las que se usarán para probar casos de uso.
Puesto que los sistemas software son cada vez más grandes y complejos, asegura que las pruebas tienen cada día más importancia. Esto es porque, además de garantizar la calidad del mismo y la satisfacción de los requisitos, ahorran tiempo y recursos en el desarrollo al localizar el mayor número de deficiencias lo antes posible.

También afirma que la comprobación del software consiste en la **verificación dinámica** (implica siempre la ejecución del programa) del comportamiento de un programa en un conjunto finito (está limitada) de casos de prueba (no se ejecuta de cualquier forma). Los casos de prueba buscan errores, e interesan el menor número de casos que encuentren el mayor número de errores.

Como elementos de una prueba se tiene:

1. Valores de prueba.
2. Acciones.
3. Resultado esperado.

Las características de una buena prueba son:

- Alta probabilidad de encontrar un fallo (Cuanto más, mejor).
- Nada redundante (si funciona, no lo volvemos a probar).
- La mejor entre las suyas (si tenemos opciones, que sea la mejor).
- Complejidad media (muy sencilla no aporta y muy compleja no sabemos lo que falla).

Los distintos estados por los que pasa el proceso de pruebas son:

1. Objetivos de prueba.
2. Diseño de casos de prueba.
3. Codificación de casos de prueba.
4. Ejecución.
5. Análisis de resultados.

También matiza que un programa que pasa todos sus casos de prueba no es un programa sin errores, puesto que las pruebas solamente pueden encontrar los errores que buscan. **De aquí la importancia del buen diseño de las pruebas**.

9.1.1. Pruebas de software

El objetivo de una prueba es presentar información sobre la calidad del producto a los responsables del mismo, por lo tanto, extrapolado al software puede decirse que, los objetivo de las pruebas de calidad son:

- encontrar defectos/bugs.
- aumentar la confianza en el nivel de calidad.
- facilitar información para la toma de decisiones.
- evitar la aparición de defectos.

Estos objetivos dependen del **contexto** en el que se desarrolle el software: context-driven testing [CDT01], concepto inicialmente desarrollado por James Bach [BaJ], Brian Marick [MaB], Bret Pettichord [PeB] y Cem Kaner [KaC].

Para ser objetivos en las pruebas, se aconseja separar el ambiente de testing del de desarrollo del software. Pero se matiza que no existen las "mejores prácticas". Lo que para una situación puede ser idílico, en otra puede estar condenado al fracaso.
Por eso la importancia de seleccionar los elementos condicionantes de las pruebas a realizar y usarlos de la manera más eficientes según el contexto de cada proyecto (como por ejemplo: actividades, técnicas, documentación, enfoques, ...).

Las pruebas pueden agruparse según sean:

- Estáticas:
  Pruebas sin ejecución del código de la aplicación (revisión de documentos).
- Dinámicas:
  Pruebas que requieren la ejecución de la aplicación para poder llevarse a cabo. Permiten usar técnicas de "caja-negra" y "caja-blanca" con más amplitud. La ejecución de la aplicación permite medir con más precisión el comportamiento de la aplicación desarrollada.

Los tipos de pruebas más destacados son los siguientes:

- De compatibilidad:
  Se comprueba el funcionamiento del software desarrollado en distintas plataformas (como por ejemplo: sistemas operativos, navegadores, redes, hardware, ...).
- De regresión:
  Se comprueba el funcionamiento del software desarrollado en evoluciones o cambios funcionales para asegurar que los casos de prueba ya probados exitosos sigan siéndolo. Este tipo de pruebas deberían ser **automáticas** para reducir tiempo y esfuerzo en su ejecución.
- De integración:
  Se comprueba el funcionamiento de la comunicación entre los componentes de un mismo sistema, entre sistemas o entre hardware y software.

Según Jorge Hernan Abad Londoño [AbJ] las pruebas de software pueden ordenarse por niveles de profundidad [AbJ05]:

1. Pruebas Unitarias.
2. Pruebas de Integración.
3. Pruebas de Sistemas.
4. Pruebas de Aceptación de Sistemas a Medida.

Se adjunta un esquema en la Figura 2 del anexo: Esquema de los Niveles de Pruebas del Software.

9.1.1.1. Pruebas Unitarias

Las pruebas unitarias son una forma de comprobar el correcto funcionamiento de un módulo de código de acuerdo con las especificaciones. Sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

- **¿Cuándo?**
  - Durante la CONSTRUCCIÓN de un sistema.
- **¿Objetivo?**
  - Probar DISEÑO+COMPORTAMIENTO de los componentes.
- **Metodología:**
  - Particionar los módulos a probar en *unidades lógicas* de fácil comprobación.
  - Definir los casos de prueba de cada *unidad lógica* (pruebas de caja blanca).

        - los casos de prueba deben recorrer todos los caminos de ejecución posibles dentro del código a probar (deben construirse con acceso al código fuente de la unidad a probar).
        - deben considerarse:
            - Rutinas de excepción,
            - Rutinas de error,
            - Manejo de parámetros,
            - Validaciones,
            - Valores válidos,
            - Valores límites,
            - Rangos y
            - Mensajes posibles.
- **Procedimiento**
    - Comparar el resultado esperado y el obtenido.
    - Reportar los errores existentes.
- **Finalización**
    - Ejecución de todas las pruebas planeadas.
    - Identificación y consideración de todos los defectos.
- **Técnicas**
    - análisis de caminos,
    - partición de categorías,
    - mutaciones,
    - algoritmos genéricos,
    - análisis de valores límite,
    - gráficos causa-efecto,
    - ...

9.1.1.2. Pruebas de Integración

Las pruebas de Integración engloban los siguientes tipos de tests:

1. Pruebas de Integración.
2. Pruebas de Regresión.
3. Pruebas de Humo.

9.1.1.2.1. Pruebas de Integración

Las pruebas de integración son una forma de comprobar el correcto funcionamiento de todos los elementos unitarios que componen un proceso, hecha conjuntamente, una única vez. Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos, combinándolos y probados como un grupo.

- **¿Cuándo?**
    - Durante la CONSTRUCCIÓN de un sistema.
- **¿Objetivo?**
    - Comprobar UNIÓN+FUNCIONALIDAD de los componentes a través de sus interfaces.
- **Metodología:**
    - Describir el *modo de verificar* que las interfaces entre las componentes de software funcionan bien.
    - Determinar el *modo de cargar* la base de datos de prueba.
    - Determinar el *modo de ascender* en los niveles de integración.
    - Determinar las *acciones a tomar* al descubrir problemas.
- **Procedimiento:** (en cada paso de la metodología)
    - Comparar el resultado esperado y el obtenido.
    - Utilizar la técnica *top-down.*
    - Utilizar la técnica *bottom-up.*
- **Finalización**
    - Ejecución de todas las pruebas planeadas.
    - Identificación y consideración de todos los defectos.
- **Técnicas**
    - top-down.

        - Se empieza probando que los módulos superiores llaman a los de nivel inferior correctamente, con los parámetros correctos.
        - Ventaja: las interfaces se prueban pronto y mucho.
    - bottom-up
        - Se empieza probando que los módulos inferiores llaman a los de nivel superior correctamente, con los parámetros correctos.
        - Ventaja: se puede paralelizar el desarrollo.
        - Inconveniente: dificulta la planificación y gestión.

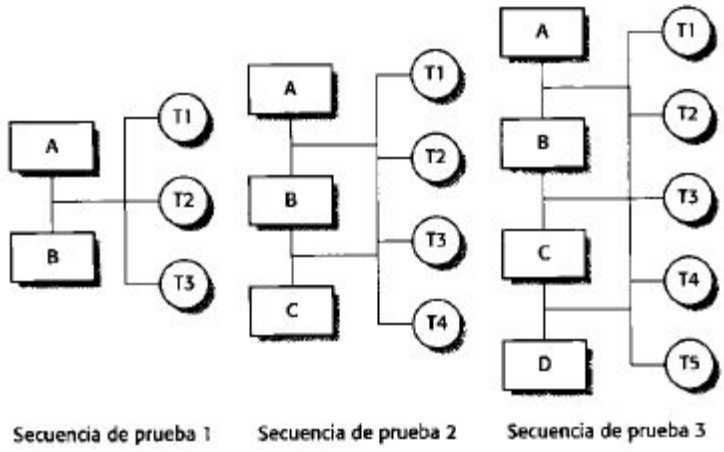


Imagen 7: Secuencia de pruebas de integración

9.1.1.2.2. Pruebas de Regresión

Las pruebas de regresión son una forma de comprobar el correcto funcionamiento de los elementos de un proceso tras realizar un cambio en el programa. Consisten en grabar una prueba que exponga el bug una vez localizado y corregido, y se vuelva a probar regularmente con cada cambio posterior.

- **¿Cuándo?**
    - Durante la CONSTRUCCIÓN de un sistema.
- **¿Objetivo?**
    - Comprobar la ADAPTABILIDAD del sistema a cambios recientes.
- **Metodología:**
    - Probar la nueva versión del sistema buscando efectos adversos en otras partes.
- **Procedimiento:** (nueva pasada de casos de pruebas)
    - Decidir qué casos de prueba incluir para probar el sistema eficientemente.
    - Automatizar la prueba (muy repetitiva).
        - Comprobar viejas funcionalidades.
        - Incluir las pruebas de los casos de uso que descubren defectos tempranamente.
- **Finalización**
    - Ejecución de todas las pruebas planeadas.
    - Identificación y consideración de todos los defectos.

9.1.1.2.3. Pruebas de Humo

Las pruebas de humo son una comprobación rápida del correcto funcionamiento de los elementos de un proceso, previa a la realización de una recepción formal. Consisten en validar el funcionamiento de las operaciones básicas para ver que funciona sin defectos, dando prioridad al porcentaje del total del sistema que se pone a prueba, sacrificando de ser necesario la finura de estas.

- **¿Cuándo?**

    - Durante la CONSTRUCCIÓN de un sistema.
- **¿Objetivo?**
    - Detectar errores en entregas tempranas.
    - Probar el sistema constantemente.
    - Garantizar poco esfuerzo en la INTEGRACIÓN final del sistema.
    - Asegurar los resultados de las pruebas UNITARIAS.
    - Reducir los riesgos y aumentar la calidad.
- **Metodología:**
    - Comprobar el sistema de extremo a extremo de forma NO EXHAUSTIVA.
- **Procedimiento:** (nueva pasada de casos de pruebas)
    - Integrar todo el sistema periódicamente (máximo 1 semana).
    - Hacer los casos de prueba de los casos de uso acabados ese día sumados a los de los días anteriores (recomendable automatización).
    - Buscar errores eficientemente.
- **Finalización**
    - Ejecución de todas las pruebas planeadas.
    - Identificación y consideración de todos los defectos.

9.1.1.3. Pruebas de Sistemas

Las pruebas de Sistemas engloban los siguientes tipos de tests:

1. Pruebas de Sistemas.
2. Pruebas de Implantación.
3. Pruebas de Aceptación.

9.1.1.3.1. Pruebas de Sistemas

Las pruebas de sistemas son una forma de comprobar que el sistema o programa cumple los objetivos originales; requerimientos funcionales y no funcionales.

- **¿Cuándo?**

  Durante la CONSTRUCCIÓN de un sistema.
- **¿Objetivo?**

  Probar A FONDO (FUNCIONALIDAD+INTEGRIDAD) el sistema en un entorno lo más parecido al entorno final de producción.
- **Metodología:**
    1. Verificar el ingreso, procesamiento y recuperación apropiados de datos.
    2. Verificar la implementación apropiada de las reglas de negocio.
    3. Determinar las pruebas de sistema (usabilidad, volumen, desempeño, etc...) que asegurarán que la aplicación alcance los objetivos de negocio.
    4. En **sistemas web** se recomienda llevar a cabo, al menos:
        - Humo.
        - Usabilidad.
        - Rendimiento.
        - Funcionalidad.
- **Procedimiento:** (en cada caso de uso, flujo básico o función)
    1. Comparar el resultado esperado y el obtenido.
    2. Aparición de mensajes de error en el momento adecuado (datos inválidos).
    3. Utilizar cada regla de negocios adecuadamente.
- **Finalización**
    - Ejecución de todas las pruebas planeadas.
    - Identificación y consideración de todos los defectos.
- **Técnicas** (probarlo como lo hará el usuario final) (**rojo** = ampliadas en otro apartado)
    - pruebas funcionales:
        - **pruebas de integración.**
        - **pruebas de entrega/funcionales**: comportamiento del sistema (proceso de pruebas de caja negra: estudiando sus entradas y salidas relacionadas).
    - pruebas no funcionales (objetivos):
        - **pruebas de comunicación**: funcionamiento de las interfaces entre componentes (y entre hombre/máquina).

- **pruebas de recuperación**: recuperación del sistema ante fallos y reanudación del procesamiento en un tiempo determinado.
- **pruebas de múltiples sitios**: configuraciones y comunicaciones de datos entre múltiples sitios.
- **pruebas de volumen**: funcionamiento del sistema con cargas de trabajo esperadas pero con grandes volúmenes de datos.
- **pruebas de sobrecarga/stress**: funcionamiento del sistema en el umbral límite de recursos, sometiendolo a cargas masivas. Busca los puntos en los que el sistema empieza a funcionar por debajo de los requisitos establecidos.
- **pruebas de carga/tensión**: someter al sistema a grandes cargas o tensiones (operaciones / segundo).
- **pruebas de disponibilidad de datos**: recuperación física y lógica del sistema sin comprometer la integridad de los datos.
- **pruebas de facilidad de uso**: adaptabilidad del sistema a las necesidades de los usuarios.
- **pruebas de operación**: implementación de los procedimientos de operación (planificación y control de trabajos, arranque y rearranque del sistema).
- **pruebas de entorno**: interacciones del sistema con otros sistemas del mismo entorno.
- **pruebas de seguridad**: mecanismos de protección integrados en el sistema.
- **pruebas de usabilidad**: problemas de usabilidad (muy subjetivos).
- **pruebas de rendimiento**: problemas de velocidad y otros atributos de calidad (escalabilidad, fiabilidad y uso de lo srecursos) de un sistema en la realización de tareas.
- **pruebas de implantación**
- **pruebas de desempeño**: desempeño del software en tiempo de ejecución dentro del contexto de un sistema integrado.
- **pruebas de resistencia**: afrontamiento de situaciones anormales; ejecución del sistema para que requiera una cantidad, frecuencia o volumen anormal de recursos.
- **pruebas de almacenamiento**: ocultación de los objetivos de almacenamiento.
- **pruebas de configuración**: funcionamiento con cada tipo de dispositivo de hardware y la configuración mínima y máxima.
- **pruebas de instalación**: problemas de usabilidad durante la instalación (pueden hacer que el cliente busque otro producto o desconfíe en la validez de la aplicación).
- **pruebas de documentación**: exactitud y claridad de la documentación del usuario.

9.1.1.3.2. Pruebas de Implantación

Las pruebas de implantación se deben llevar a cabo como sigue:

- **¿Cuándo?**

  Durante la IMPLANTACIÓN en un entorno de PRODUCCIÓN.
- **¿Objetivo?**

  Comprobar el FUNCIONAMIENTO (RENDIMIENTO, COPIAS DE SEGURIDAD, ...) en el entorno real de producción.
- **Técnicas**

  Análisis de calinos.

9.1.1.3.3. Pruebas de Aceptación

Las pruebas de aceptación se deben llevar a cabo como sigue:

- **¿Cuándo?**

  TRAS LA IMPLANTACIÓN en un entorno de producción.
- **¿Objetivo?**

  Comprobar que se cumplan los REQUISITOS y permitir que los usuarios den el VISTO BUENO final.
- **Técnicas**
  a. Pruebas alfa: detectar errores en el sistema bajo un ambiente controlado; se llevan a cabo en el lugar en el que se desarrolla el software en presencia del desarrollador.
  b. Pruebas beta: validación del software en un ambiente real; se llevan a cabo por personas en su área de trabajo real sin supervisión.

9.1.1.4. Pruebas de Aceptación de Sistemas a Medida

Las pruebas de Aceptación de Sistemas a medida engloban los siguientes tipos de tests:

1. Pruebas del Ciclo de Negocio.
2. Pruebas de GUI.
3. Pruebas de Configuración.
4. Pruebas de Estilo.
5. Pruebas de Aceptación.
6. Pruebas de Instalación.
7. Pruebas Funcionales.
8. Pruebas de Documentación y Procedimiento.
9. Pruebas de Usabilidad.
10. Pruebas de Campo.

9.1.1.4.1. Pruebas del Ciclo de Negocio

Las pruebas del ciclo de negocio son una forma de comprobar el sistema a lo largo de todo un ciclo completo de negocio, generalmente un año fiscal u otra unidad de tiempo similar. Se acompaña con todos los procedimientos necesarios para evaluar en poco tiempo lo que normalmente ocurre a lo largo de un periodo extenso de tiempo [Ga108].

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema.
- **¿Objetivo?** [LeG12]
  Comprobar que el sistema funciona de acuerdo con el modelo de negocio emulando todos los EVENTOS EN EL TIEMPO y en función del tiempo.
- **Metodología:**
  a. Emular las actividades ejecutadas en el ciclo de negocio a través del tiempo.
  b. Identificar un periodo (por ejemplo un año fiscal).
  c. Ejecutar las transacciones y actividades que podrían ocurrir en el periodo identificado, incluyendo todos los ciclos y eventos diarios, semanales y mensuales con datos sensibles.
- **Procedimiento** (en cada caso de uso, flujo básico o función con datos válidos e inválidos)**:**
  a. Incrementar el número de veces que se ejecuta una función para simular distintos usuarios en un periodo determinado.
  b. Probar fechas y periodos de tiempo inválidos en todas las fechas o funciones que involucren tiempos.
  c. Ejecutar en el tiempo apropiado las funciones que ocurren en un periodo de tiempo.
  d. Recibir resultados esperados al usar datos válidos.
  e. Recibir mensajes de error esperados al usar datos inválidos.
  f. Aplicar adecuadamente cada regla de negocio.
- **Finalización:**
  a. Ejecución de todas las pruebas planeadas.
  b. Identificación y consideración de todos los defectos.

9.1.1.4.2. Pruebas de GUI

Las pruebas de GUI[17] son una forma de evaluar el sistema mediante pruebas con los propios usuarios, para medir la capacidad del sistema de cumplir el propósito para el que fue diseñado.

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema.
- **¿Objetivo?**
  Comprobar que la NAVEGACIÓN DEL SISTEMAa a través de los objetos refleja las funcionalidades del negocio y los requisitos, usando los modos de acceso (tabuladores, movimientos del ratón, atajos del teclado, etc).
- **Metodología:**
  1. Asegurar que la interfaz tiene una navegación apropiada a través de las diferentes funcionalidades.
  2. Asegurar que los objetos de la interfaz se encuentren en los estándares de la indústria.
- **Procedimiento:**
  1. Probar la creación y modificación de cada ventana para verificar la navegación y el estado de los objetos.

[17] Interfaz Gráfica de Usuario: del inglés, Graphical User Interface

- **Finalización**
    1. Verificación y comparación total de cada ventana con otras similares del mercado, logrando una buena aceptación dentro del estándar.

9.1.1.4.3. Pruebas de Configuración

Las pruebas de configuración son una forma de comprobar el sistema usando todas las configuraciones de software y hardware que el mismo soporta. Se valida también que el proyecto actual sea capaz de soportar diferentes tecnologías hardware (por ejemplo impresoras, interfaces, ...) [PC].

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema.
- **¿Objetivo?**
  Comprobar que el cliente del sistema funciona en las ESTACIONES DE TRABAJO recomendadas.
- **Metodología:**
    1. Probar el programa con cada tipo de dispositivo, con las configuraciones mínima y máxima posibles.
- **Procedimiento:**
    1. Usar los scripts de Integración y Pruebas del Sistema.
    2. Incluir la apertura o cierre de varias aplicaciones (similares a las que se esten probando) como parte de la prueba.
    3. Ejecutar algunas transacciones para simular actividades cotidianas del usuario, dentro y fuera de las actividades que interactúan con la BD.
    4. Repetir los pasos minimizando la cantidad de memoria convencional disponible en los clientes.
- **Finalización:**
    1. Ejecución sin fallos de cada combinación de transacciones de las aplicaciones que interactúan con la BD.

9.1.1.4.4. Pruebas de Estilo

Las pruebas de estilo son una forma de comprobar los formatos de la interfaz (por ejemplo ventanas, colores corporativos, tipos de letra, etc...):

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema.
- **¿Objetivo?**
  Comprobar que el sistema sigue los ESTÁNDARES DE ESTILO propios del cliente.
- **Metodología:**
    1. Comprobar que el formato visual del sistema es el especificado por el cliente.
- **Procedimiento:**
    1. Navegar por el sistema verificando si se cumplen los estándares de GUI del cliente.
    2. Validar cada objeto gráfico con el manual de estilos del cliente.
- **Finalización:**
    1. Ejecución de todas las pruebas planeadas.
    2. Identificación y consideración de todos los defectos.

9.1.1.4.5. Pruebas de Aceptación

Las pruebas de aceptación [PDA] son una forma de comprobar el grado de calidad del software con relación a todos los aspectos relevantes para que el uso del producto esté justificado.

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema, en un entorno de PRE-PRODUCCIÓN.
- **¿Objetivo?**
  Comprobar que el sistema se ajusta a los requisitos fijados y, por lo tanto, está listo para ser implantado para el uso operativo en el entorno del usuario. Esto hará que sea aceptado o rechazado por el cliente.
- **Metodología** (desarrollada y ejecutada por el cliente o un especialista)**:**
    1. Comprobar que los requisitos incluyen la documentación y procesos de negocio.
    2. Llevar a cabo un subconjunto de las Pruebas de Sistema para comprobar si el producto está listo par el uso operativo.

3. Determinar si se acepta o rechaza el sistema.

- **Procedimiento:**
  1. Realizar los documentos de planes de prueba de aceptación y especificación de los mismos, basados en los criterios de aceptación del cliente.
  2. Planificar, organizar y formalizar los casos de prueba de aceptación de modo que determinen el cumplimiento de los requisitos del sistema. Para ello se precisan los documentos de:
     a. Especificación de requisitos del sistema.
     b. Manual de usuario.
     c. Manual de administrados.
  3. Realizar las Pruebas de Estilo.
- **Finalización:**
  1. Ejecución de todas las pruebas planeadas.
  2. Identificación y consideración de todos los defectos.

9.1.1.4.6. Pruebas de Instalación

Las pruebas de instalación son una forma de comprobar la instalación del sistema en el cliente:

- **¿Cuándo?**
  TRAS la INSTALACIÓN de un sistema.
- **¿Objetivo?**
  Comprobar que, en cada cliente, el sistema se INSTALA APROPIADAMENTE según sea:
  - nueva instalación: máquina nueva a la que nunca se le ha instalado un sistema.
  - actualización: máquina previamente instalada con el sistema.
  - downgrade: máquina previamente instalada con un sistema más actual.
- **Metodología:**
  1. Asegurar que el sistema puede ser instalado en todas las configuraciones posibles (por ejemplo nuevas instalaciones, actualizaciones, instalaciones completas o personalizadas) y bajo condiciones normales o anormales (por ejemplo espacio insuficiente en disco, falta de privilegios para algunas tareas, ...).
  2. Correr un número significativo de Pruebas de Funcionalidad para verificar que, una vez instalado, el sistema opera correctamente.
- **Procedimiento:**
  1. Diseñar los scripts para validar las condiciones de la máquina a instalar.
  2. Realizar la instalación.
- **Finalización:**
  1. Ejecución de las transacciones de la aplicación sin fallos.

9.1.1.4.7. Pruebas Funcionales

Las pruebas funcionales son una forma de comprobar la ejecución, revisión y retroalimentación de las funcionalidades diseñadas para el software. Se prueba y valida que el software hace lo que debe y, sobre todo, lo que se ha especificado. Estas son, por lo tanto, el **tipo de pruebas que se llevarán a cabo en la aplicación a desarrollar**.

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema.
- **¿Objetivo?**
  Comprobar que el sistema cumple los REQUISITOS FUNCIONALES:
  - Navegación.
  - Entrada de datos.
  - Procesamiento.
  - Obtención de resultados.
- **Metodología:**
  1. Enfocar las pruebas funcionales en los requisitos funcionales, basándolas en los Casos de Uso y las reglas del negocio:
     a. Verificar la aceptación de datos.
     b. Verificar el procesamiento, la recuperación y la implementación adecuada de las reglas de negocio.
  2. Verificar la aplicación y sus procesos internos analizando la salida (resultados) mediante la interacción vía GUI (técnica de caja negra).

- **Procedimiento:**
    1. Ejecutar cada Caso de Uso, flujo de caso de uso o función usando datos válidos e inválidos.
    2. Verificar:
        a. Los resultados esperados ocurren al usar datos válidos.
        b. Los mensajes de error y precaución se despliegan apropiadamente al usar datos inválidos.
        c. Cada regla de negocio se aplica apropiadamente.
- **Finalización:**
    1. Ejecución de todas las pruebas planeadas.
    2. Identificación y consideración de todos los defectos.

9.1.1.4.8. Pruebas de Documentación y Procedimiento

Las pruebas de documentación y procedimiento son una forma de comprobar que el usuario dispone de todo aquello que necesita para la instalación, el aprendizaje y el uso del producto (guías de instalación, del usuario, de mensajes, manuales de referencia, etc).

- **¿Cuándo?**
  Durante la CONSTRUCCIÓN de un sistema.
- **¿Objetivo?**
  Evaluar la DOCUMENTACIÓN del usuario.
- **Metodología:**
    1. Determinar si el manual de procedimientos trabaja correctamente (como parte integral del sistema) evaluando la exactitud y claridad de la documentación del usuario.
    2. Probar cualquier procedimiento humano tal como lo llevan a cabo:
        a. el operador,
        b. el administrador de la BD,
        c. o el usuario final.
- **Procedimiento:**
    1. Verificar la documentación del proyecto comparándola con las funcionalidades del sistema y su configuración física.
- **Finalización:**
    1. Ejecución de todas las pruebas planeadas.
    2. Identificación y consideración de todos los defectos.

9.1.1.4.9. Pruebas de Usabilidad

Las pruebas de usabilidad son una forma de evaluar el sistema mediante pruebas con los propios usuarios, para medir la capacidad del sistema de cumplir el propósito para el que fue diseñado.

- **¿Cuándo?**
  Durante la IMPLANTACIÓN de un sistema.
- **¿Objetivo?**
  Comprobar la usabilidad del sistema.
- **Metodología:**
    1. Determinar lo bien que el usuario podrá usar y entender la aplicación.
    2. Identificar las áreas de diseño que hacen que el sistema sea difícil de usar.
    3. Detectar los problemas de conveniencia y practicidad del sistema desde el punto de vista del usuario.
- **Procedimiento:**
    1. Verificar que la aplicación no padece de:
        a. Sistema demasiado complejo y difícil de usar.
        b. Dificultad de instalación y comprensión del sistema.
        c. Recuperación de errores pobre y mensajes de error sin significado.
        d. Sintaxis de comandos difícil de aprender y recordar.
        e. Obligación al usuario a recordar formatos y secuencias fijas.
        f. Procedimientos poco simples ni obvios.
        g. Carencia de instrucciones de ayuda por ordenador y manual pobre.
        h. Calidad y apariencia pobres en diagramas, pantallas, reportes y gráficos.
        i. Carencia de herramientas de construcción adecuadas y requerimiento de múltiples comandos.

2. Crear casos de prueba para comprobar que se puede operar en el sistema correctamente.

- **Finalización:**
  1. Ejecución de todas las pruebas planeadas.
  2. Identificación y consideración de todos los defectos.

9.1.1.4.10. Pruebas de Campo

Las pruebas de campo son una forma de comprobar el sistema con prototipos del software y grupos formados por los usuarios finales para obtener retroalimentación y, donde sea necesario, realizar correcciones y mejoras al software basado en ambientes virtuales [Fue03].

- **¿Cuándo?**
  Durante la PRODUCCIÓN de un sistema.
- **¿Objetivo?**
  Validar el producto contra sus especificaciones originales y encontrar errores corriendo el sistema en el ambiente real.
- **Metodología:**
  1. Realizar un subconjunto válido de pruebas de sistema.
- **Procedimiento:**
  1. Determinar que las pruebas de sistema serán ejecutadas para validar el sistema en producción.
- **Finalización**
  1. Ejecución de todas las pruebas planeadas.
  2. Identificación y consideración de todos los defectos.

9.2. Herramientas de testeo

Según Etienne Wenger [Wen01], "*los trabajadores organizan su vida con sus compañeros y clientes inmediatos para llevar a cabo su trabajo. Con ello desarrollan o mantienen un sentido de sí mismos con el que poder vivir, divertirse un poco y cumplir las exigencias de sus patrones y clientes. Independientemente de cuál pueda ser la descripción oficial de su trabajo, crean una práctica para hacer lo que es necesario hacer. Aunque los trabajadores puedan estar contratados por una gran institución, en la práctica diaria trabajan con - y, en cierto sentido, para - un conjunto mucho más pequeño de personas y comunidades*."

"Por lo tanto, las comunidades de prácticas (CP) son grupos de personas que tienen en común un interés o una pasión por alguna actividad y que gracias a la interacción regulada aprenden a mejorarla. Una CP no es simplemente una comunidad de interés (por ejemplo, personas que tienen afición por determinados tipos de película), los miembros de una CP son profesionales que desarrollan un abanico común de recursos (experiencias, anécdotas, herramientas, formas de resolver problemas recurrentes). En pocas palabras, es una práctica compartida."

No es de extrañar, pues, que grandes entidades como con las que trabaja Everis, empiecen a desarrollar portales web con capa social para potenciar y mejorar la comunicación entre sus trabajadores y colaboradores. Puesto que estos portales se integran con las funcionalidades y procesos ya existentes en la empresa, la complejidad del sistema es elevada. De aquí se deduce la importancia y envergadura de su desarrollo, y desarrollo implica testeo.

9.2.1. ¿Cómo se realiza la actividad a mejorar?

Actualmente, uno de los procedimientos que se siguen en el departamento de digital channels para realizar planes de pruebas, es mediante ficheros de excel que se van rellenando a mano. Una vez desarrollada la nueva funcionalidad, se crea una plantilla con las pruebas a validar. Esto significa que se diseñan los juegos de datos de entrada, se define lo que se espera y se valida la funcionalidad desarrollada. Para garantizar que el conjunto del sistema funciona bien, antes de enviar los cambios al cliente (*subir* a PRO) se realizan las pruebas donde, para un input concreto, el sistema debe devolver el resultado esperado.

Una vez *subido* a PRO, se vuelve a llevar a cabo el mismo proceso con el mismo plan de pruebas de los desarrolladores (PRE) para comprobar que todo sigue funcionando en este entorno.

9.2.2. ¿Cómo se realizan actividades semejantes en otros ámbitos?

La comprobación del software puede llevarse a cabo de forma manual o automática. Todo depende del producto, del proyecto y de la empresa.

En función del producto (tecnología usada), hay que estudiar las herramientas de automatización existentes. Estas ayudan a ver (con sus interfaces) el nivel de dificultad para automatizar las pruebas. En casos como el estudiado en los que el producto requiere trabajar en múltiples plataformas (distintos navegadores y entornos), se puede ahorrar tiempo de desarrollo reutilizando

las pruebas automatizadas: realizando las mismas pruebas escogidas a través de los parámetros de configuración del fichero de propiedades.

En función del proyecto, el número de iteraciones del desarrollo y su frecuencia pueden variar. Como remarca S. Mendoza [Men11], las iteraciones no son exclusivas de la fase de desarrollo, puesto que una vez liberado el software, puede haber peticiones correctivas y evolutivas. También hay casos en los que el ritmo y la calidad del proyecto vienen determinados por las exigencias del mercado. Esto sirve de guía sobre cuánto se usarán las pruebas automatizadas.

En función de la empresa, la automatización puede llevarse a cabo desde el principio del proyecto o una vez finalizado. Generalmente, las empresas pequeñas y nuevas no pueden afrontar una automatización de los procesos de pruebas por temas económicos y temporales. Aunque hay empresas que, teniendo la capacidad, van posponiendo la automatización hasta llegar a sacar versiones prácticamente sin testear [Far11]. Los principales motivos suelen ser el tamaño del producto, la alta frecuencia de salida de nuevas versiones y la pluralidad de versiones para distintos clientes o plataformas, entre otros. Para que el proyecto de automatización tenga éxito, se precisa el apoyo de la capa de gerencia y del equipo de desarrollo. De aquí la importancia de hacerles entender sus beneficios.

Por lo tanto, para decidir el momento exacto en el que automatizar un proceso de testing, se deben tener en cuenta las variables anteriores. Esto es así puesto que *muy pronto* puede implicar hallarse con una inmadurez a nivel de equipo, y *demasiado tarde* puede conllevar que algunos cambios del desarrollo se detecten muy tarde y haya una *regresión*[18] al final. Es evidente que estos pueden afectar al éxito del producto.

Por ello hay que estudiar desde el comienzo del proyecto si se está en el momento correcto para llevar a cabo la automatización de sus procesos de pruebas.

9.2.3. Características de la tecnología a utilizar

El proceso de automatización se debe apoyar en herramientas de testing. La gama de herramientas a utilizar varía en función de los tests a hacer. En esta clase de proyectos, el tipo de tests que tienen más sentido son:

- Tests de gestión de pruebas.
- Tests de pruebas funcionales.
- Tests de carga y rendimiento.

A continuación se listan algunas de las mejores herramientas open-source para desarrollar tests de gestión de pruebas, para llevar a cabo pruebas funcionales y para probar la carga y rendimiento de una web [TS13]:

Pruebas funcionales	**Gestión de pruebas**	**Carga y rendimiento**
Selenium	Bugzilla	FunkLoad
SoapUI	**FitNesse**	FWPTT load testing
Watir (web en Ruby)	qaManager	loadUI

[18] Descubrir las causas de nuevos errores, carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, inducidos por cambios recientemente realizados en partes de la aplicación que anteriormente al citado cambio no eran propensas a este tipo de error.

WatiN (web en .NET)	qaBook	**JMeter**
Canoo WebTest	RTH (open source)	
Solex	Salome-TMF	
Imprimatur	Squash TM	
SAMIE	Test Enviroment Toolkit	
ITP	**TestLink** (open source)	
WET	Testitool	
WebInject	XQual Studio	
	Radi-testdir	
	Data Generator	

Tabla 12: Herramientas de pruebas funcionales, de gestión de pruebas y de carga y rendimiento

En negrita se muestran las herramientas más populares según [OST03] en el momento en el que se ha llevado a cabo el estudio.
Las principales características de las herramientas de ***pruebas funcionales*** son [OST03]:

- **Selenium**: herramienta de testeo para probar aplicaciones web basadas en navegadores. Puede usarse para llevar a cabo tests funcionales de compatibilidad (alta compatibilidad entre navegadores) y de regresión.
    - Requisitos: Mac, Windows o Linux.
- **Watir** (Web Application Testing in Ruby): herramienta de testing funcional para aplicaciones web. Apoya las pruebas ejecutadas en la capa del explorador web usando el explorador e interactuando con los objetos de la página. Usa el lenguaje de scripting Ruby.
    - Requisitos: Windows (Internet Explorer).
- **WatiN** (Web Application Testing in dotNet): Está inspirado en Watir y permite las aplicaciones de testeo web expresadas en cualquier lenguaje .Net, a través de Internet Explorer en una plataforma Windows.
    - Requisitos: Windows.
- **Canoo WebTest**: es un marco de trabajo open-source, construido sobre HttpUnit usado para llevar a cabo tests funcionales de páginas web. Permite definir los tests tanto en XML como en Objetivos Ant (Ant Targets).
    - Requisitos: JDK 1.2 y ANT v1.3.

- SoapUI: aplicación de escritorio basada en swing (de Java) para inspeccionar, invocar y llevar a cabo tests funcionales de servicios web sobre HTTP. Está dirigido mayoritariamente a desarrolladores/testers que provean y/o consuman servicios web (java, .net, etc). Los tests funcionales pueden llevarse a cabo interactivamente en soapUI o en un proceso de integración continua usando el plugin maven de soapUI.
    - Requisitos: Java 1.5.
- Solex: conjunto de plugins de Eclipse que permiten llevar a cabo tests de stress y de no regresión a servers de aplicaciones web. Los scripts de prueba se graban de un explorador de internet gracias a un proxy web incorporado.
    - Requisitos: Eclipse 2.1 o superior.

- Imprimatur: herramienta de pruebas funcionales para aplicaciones web. Los tests se describen en un fichero XML simple. Manipula métodos HTTP, autenticación y subidas de ficheros. Las respuestas pueden validarse usando expresiones regulares.
    - Requisitos: Java.
- Samie: módulo de Perl que permite al usuario ejecutar tests automatizados para sus aplicaciones web.
    - Requisitos: Windows NT/2000.
- ITP: instrumento de pruebas de aplicaciones web muy ligero y poderoso. Scripts de prueba escritos en XML. No precisa programación ni llevar a cabo cambios en la aplicación web. Apoya sesiones/cookies y fechas en formato POST. Basado en línea de comandos para la integración con otras herramientas. Útil para llevar a cabo tests de regresión y pruebas de humo.
    - Requisitos: Independiente del SO.
- WET: herramienta open-source de pruebas automatizadas en web que usa Watir como librería para manejar las páginas web. No precisa instalar ni conocimientos sobre Watir. Maneja directamente un explorador Internet Explorer, por lo que la prueba automatizada se lleva a cabo del mismo modo en que un usuario navegaría por las páginas web. Gracias al uso de puntos de control, permite hacer varias comprobaciones como parte del proceso de pruebas.
    - Requisitos: Windows 98/ME/2000 SP3/XP SP2/Server 2003.
- WebInject: herramienta gratuita para llevar a cabo pruebas automatizadas de aplicaciones web y servicios. Puede usarse para probar cualquier componente del sistema individualmente con interfaz PHP, y como instrumento de pruebas para crear un conjunto de pruebas funcionales y de regresión automatizadas.
    - Requisitos: Windows, OSX, Linux.

Las principales características de las herramientas de ***gestión de pruebas*** son [OST03] :

- **Fitnesse**: herramienta de testeo y documentación colaborativa. Permite crear documentos, especificar tests y ejecutarlos de forma sencilla.
    - Requisitos: Mac, Windows, POSIX.
- **TestLink**: sistema de gestión y ejecución de tests basado en web. Incluye especificación de tests, planeamiento, informes, seguimiento de requisitos y colaboración con los controladores de errores (bug trackers) más conocidos.
    - Requisitos: Apache, MySQL, PHP.

- qaManager: aplicación basada en web usada para hacer seguimiento de ingeniería y comunicar a clientes el lanzamiento de nuevos proyectos software la asignación de recursos e información relacionada.
    - Requisitos: independiente de la plataforma.
- RTH: herramienta basada en web diseñada para gestionar requisitos, tests, resultados de tests y defectos a través del ciclo de vida de la aplicación. Proporciona un acercamiento estructurado al testeo de software e incrementa la visibilidad de los procesos de testing creando un repositorio común para todos los activos de prueba que incluyen requisitos, casos de testeo, planes de testeo y resultados de los tests.
    - Requisitos: Cualquier Windows de 32-bits (95-XP), POSIX, IBM AIX.
- Salome-TMF: herramienta de gestión de tests independiente que ayuda a gestionar el proceso de testeo creando tests, ejecutándolos a mano o automáticamente, haciendo un seguimiento de los resultados, gestionando los requisitos y defectos y produciendo documentación HTML. Es compatible con Junit, Abbot y Beanshell para definir tests automáticos. Es compatible con Bugzilla y Mantis para gestionar los defectos. Puede extenderse con plug-ins según convenga.

    - Requisitos: Java.
- Squash TM: gestor de repositorios test del kit de herramientas open-source de Squash. Permite gestionar requisitos, casos de testeo y campañas de ejecución en un contexto multiproyecto.
    - Requisitos: Windows, Mac, Linux.
- Test Environment Toolkit: versión open-source de TETware, un marco de trabajo de testeo multi-plataforma para gestionar los bancos de pruebas (administración, informes y secuenciación de tests).
    - Requisitos: Linux, UNIX.
- Testitool: aplicación basada en web para planear tests QA (quality assurance). Crea un plan de testeo y lo rellena con los casos de testeo, traza los casos de prueba para los requisitos funcionales, instancia los planes de testeo, empieza ejecutando casos de prueba y marcándolos como exitosos o fallidos, genera informes en los planes de prueba, copia planes y casos de testeo y adapta instancias de planes de prueba añadiendo y eliminando casos de prueba.
    - Requisitos: Apache, MySQL, PHP.
- XQual Studio: aplicación totalmente gráfica y gratuita de gestión de tests que se encarga del ciclo completo de los proyectos de tests QA: usuarios, requisitos, especificaciones, SUITs, proyectos de desarrollo, tests, planes de prueba, informes de las pruebas, campañas de testeo y defectos. Al usar una Base de Datos MySQL como almacenamiento principal, permite programar o ejecutar campañas de testeo manuales o totalmente automatizadas. La aplicación es gratuita pero no open-source, sus lanzadores (drivers para interconectar con cualquier tipo de test) están bajo licencia GPL[19].
    - Requisitos: Windows, Linux, MacOSX, Web (mediante un despliegue JavaWebstart).
- Radi-testdir: director de testeo muy ligero que soporta características de dirección de pruebas: configurar un plan de pruebas, actualizar (crear/editar) los resultados de la prueba para las tiendas de pruebas del conjunto de imágenes resultantes.
    - Requisitos: POSIX.
- Data Generator: Es un script gratuito y open-source, escrito en JavaScript, PHP y MySQL que permite generar grandes volúmenes de datos personalizados rápidamente, en una variedad de formatos para usar en software de testing, para rellenar bases de datos y mucho más.
    - Requisitos: -

Las principales características de las herramientas de ***carga y rendimiento*** son [OST03]:

- **JMeter**: aplicación Java de escritorio diseñada para cargar pruebas de comportamiento funcional y medir el rendimiento. Fue diseñada para probar aplicaciones web, pero se ha expandido a otras funciones de testing. Puede usarse para probar el rendimiento de recursos estáticos y dinámicos (archivos, Servlets, scripts de Perl, Objetos Java, Bases de Datos y consultas, servers FTP, etc). Puede usarse para simular altas cargas de un servidor, de una red o de un objeto para probar su resistencia o para analizar el comportamiento general bajo distintos tipos de carga. Puede usarse para hacer un análisis gráfico de rendimiento o para probar el comportamiento de un server/script/objeto bajo una gran carga concurrente.
    - Requisitos: Solaris, Windows (98, NT, 2000), JDK1.4 (o mayor).

- FunkLoad: testeador funcional y de carga web, escrito en Python. Sus principales casos de uso son funcionales y de regresión de proyectos web, pruebas de rendimiento al cargar la aplicación web y monitorear los servers, pruebas de carga para delatar bugs ocultos en las pruebas superficiales y tests de estrés para recargar los recursos de la aplicación web y

[19] Siglas del inglés General Public License: Licencia Pública General [Gnu84]

probar la recuperación de la aplicación. Permite escribir agentes web creando scripts de cualquier tarea web repetitiva (como comprobar si una página sigue viva).
    - Requisitos: Independiente del SO - el monitoreo es exclusivo de Linux.
- FWPTT load testing: programa open-source de pruebas de aplicaciones web. Puede grabar peticiones AJAX y normales. Ha sido probado en aplicaciones ASP.Net, pero debería funcionar con JSP, PHP u otros.
    - Requisitos: Windows.

Puesto que los portales con capa social para grandes sistemas corporativos han de ser independientes del explorador, se precisa una herramienta de pruebas funcionales con alta compatibilidad entre navegadores. Esto reduce el número de herramientas de pruebas funcionales a 3: Selenium, ITP y WebInject.

Para escoger las mejores herramientas de gestión de pruebas, nos centraremos en las que cumplen las características que el equipo de la empresa usa en la actualidad:

- Servidor Apache.
- Base de Datos MySQL.
- Lenguaje PHP.

Esto reduce el número de herramientas de gestión de pruebas a 2: TestLink y Testitool.

Si al final del proyecto se decide llevar a cabo pruebas de carga y rendimiento, consideraremos la herramienta más independiente en cuanto a SO: FunkLoad [FL05].

Poniendo foco en las herramientas de pruebas funcionales y de gestión de pruebas, consideraremos las que puedan tener una comunidad más grande en la que se pueda hallar respuesta a posibles problemas futuros. Esto equivale a las más conocidas, Selenium y TestLink. Para facilitar el proceso de testeo tanto en exploradores pesados con JS/AJAX como en exploradores ligeros pero sin JS/AJAX, se usará Mink (usa la API de Selenium entre otras y es mucho más ligero).

A modo de resumen, las herramientas a utilizar en cada tipo de prueba serán:

- En las pruebas funcionales se utilizará **Mink** [Min11] + **Selenium** [Sel04].
- En la gestión de las pruebas se usará **TestLink** [TL13].

9.3. Herramienta de testeo: Behat

Behat [Beh] es un marco de trabajo para BDD en PHP 5.3+. BDD es una forma de desarrollar software a través de la comunicación constante con los *stakeholders* mediante ejemplos, unos ejemplos acerca de cómo el software debería ayudar a *stakeholders* y desarrolladores a alcanzar sus objetivos. De este modo, se evitan los errores de requerimientos, que son los desajustes entre la especificación de requisitos de un proyecto y las necesidades y expectativas de los *stakeholders* [Tor12]. Por lo tanto, gracias al BDD, las partes interesadas entienden exactamente el modo en que quieren que una funcionalidad se comporte.

9.3.1. Instalación

Para instalar Behat con Composer [Com16], se debe navegar al directorio del proyecto y ejecutar el siguiente comando:

```
:dummy$ composer require behat/behat=~3.0.4
```

Se puede comprobar la versión de Behat instalada con el comando:

```
:dummy$ behat/behat -v
```

En nuestro caso, la versión instalada es la 3.0.15.

9.3.2. Configurar Banco de Pruebas

El modo que tiene Behat de saber encontrar y probar las funcionalidades de una aplicación son los bancos (suites). El banco por defecto indica a Behat que las funcionalidades se encuentran en la carpeta `/features` y que las ha de probar con la clase `FeatureContext`.

9.3.2.1. Inicializar Banco

Al inicializar un banco, el sistema informará de las clases que faltan para iniciar el testeo. La inicialización se lleva a cabo con el siguiente comando:

```
:dummy$ bin/behat --init
```

La primera vez que se ejecuta el comando, el sistema indica el lugar en el que se deben encontrar los archivos que necesita. En nuestro caso, se indica lo siguiente:

```
+d features                                  #para los archivos *.feature
+d features/bootstrap                        #para las clases contextuales
+d features/bootstrap/FeatureContext.php     #para las definiciones, transformaciones y hooks
```

9.3.2.2. Ejecutar Behat

La ejecución de Behat consiste en el inicio de la ejecución de los tests. Las primeras veces que se ejecute este comando, el sistema informará de los pasos que falta añadir en el archivo `features/bootstrap/FeatureContext.php`. para poder iniciar los tests. La ejecución se lleva a cabo con el siguiente comando:

```
:dummy$ bin/behat
```

Una vez ejecutado el comando, hay que introducir en el archivo mencionado anteriormente los pasos que se indican. Si no se quiere tener que copiar a mano los pasos e introducirlos en el fichero correspondiente, se puede hacer automáticamente ejecutando el siguiente comando:

```
:dummy$ bin/behat --dry-run --append-snippets
```

La estructura de los ejemplos sigue siempre el patrón `Contexto-Acción-Resultado` y se escriben con un formato especial llamado *Gherkin* [Ghe]. *Gherkin* es un lenguaje comprensible por humanos y ordenadores con el que se describen las funcionalidades, es decir, se define el comportamiento del software sin entrar en su implementación. Es un lenguaje fácil de leer, fácil de entender y fácil de escribir. Gracias a esto se alcanzan los dos objetivos de este lenguaje:

- generar una documentación (que sea fácil de interpretar por las partes interesadas),
- de forma automatizada.

Gherkin puede usarse en más de 60 idiomas [ReC16], entre los que se encuentran el Alemán, Árabe, Catalán, Emoji, Español, Gallego, Klingon, LOLCAT y Pirata [GHC16]. Por ello, *Gherkin* permite escribir historias de usuario usando palabras clave concretas de un idioma conocido por el desarrollador. Para comprobar si *Behat* y *Gherkin* están en un idioma concreto hay que ejecutar el siguiente comando en el terminal:

```
$ behat --story-syntax --lang=es        #idioma del ejemplo: español[es]
```

Hay que tener en cuenta que el idioma por defecto es el inglés (EN), por lo que si se usa otro idioma, al comienzo de los ficheros `*.feature` hay que añadir el comentario "`#language: es`" (idioma del ejemplo), por lo que el fichero quedaría como sigue:

```
# language: es
Característica: Internal operations
  In order to stay secret
  As a secret organization
  We need to be able to erase past agents' memory

  Antecedentes:
    [Dados|Dadas|Dado|Dada] there is agent A
    [Y|E] there is agent B

  Escenario: Erasing agent memory
    [Dados|Dadas|Dado|Dada] there is agent J
    [Y|E] there is agent K
    Cuando I erase agent K's memory
    Entonces there should be agent J
    Pero there should not be agent K

  Esquema del escenario: Erasing other agents' memory
    [Dados|Dadas|Dado|Dada] there is agent <agent1>
    [Y|E] there is agent <agent2>
    Cuando I erase agent <agent2>'s memory
    Entonces there should be agent <agent1>
    Pero there should not be agent <agent2>

    Ejemplos:
      | agent1 | agent2 |
      | D      | M      |
```

En nuestro caso, al llevarse a cabo en una empresa internacional, se ha decidio hacerlo en inglés, por lo que nuestros ficheros `*.feature` siguen la siguiente estructura:

```
[Feature|Business Need|Ability]: Internal operations
  In order to stay secret
  As a secret organization
  We need to be able to erase past agents' memory

  Background:
    Given there is agent A
    And there is agent B
```

```
Scenario: Erasing agent memory
  Given there is agent J
  And there is agent K
  When I erase agent K's memory
  Then there should be agent J
  But there should not be agent K

[Scenario Outline|Scenario Template]: Erasing other agents' memory
  Given there is agent <agent1>
  And there is agent <agent2>
  When I erase agent <agent2>'s memory
  Then there should be agent <agent1>
  But there should not be agent <agent2>

  [Examples|Scenarios]:
    | agent1 | agent2 |
    | D      | M      |
```

De este modo, las características contienen toda la información sobre el tipo de contenido (algo muy importante para las metodologías tipo BDD) y se le otorga a Behat la habilidad de tener características multi-idioma en un juego de pruebas.

Puesto que *Gherkin* está muy estructurado, facilita la automatización y auto-testeo de los ejemplos del comportamiento de una aplicación en desarrollo. Con solamente 5 palabras se puede empezar a usar BDD construyendo las sentencias con las que se describen las funcionalidades, por lo tanto, los ejemplos automatizados se usan para desarrollar la aplicación con TDD. Las palabras son las siguientes:

- **Feature**: es el nombre de la funcionalidad a probar. Ha de ser un título claro y explícito que se recomienda que siga la siguiente estructura:
  ```
  Como [ROL] quiero [CARACTERÍSTICA] para que [BENEFICIOS]
  ```
- **Scenario**: es la descripción de cada uno de los escenarios a probar.
- **Given**: es el contexto del escenario en el que se va a ejecutar el test. Son los pre-requisitos de los datos o el punto de ejecución del test. Contiene los pasos necesarios para dejar el sistema en el estado que se desea probar.
- **When**: son el conjunto de acciones que accionan el test. Es la interacción del usuario que acciona la funcionalidad que se quiere probar.
- **Then**: es el resultado que se pretende obtener en el test. Se observan los cambios en el sistema y se comprueba que sean los deseados.

Generalmente, para probar una determinada funcionalidad hacen falta distintos escenarios. Así es como se pasa de tener historias de usuario a tener pruebas de comportamiento automatizadas.
Por lo tanto, Behat es un ejecutable que corre desde línea de comandos para comprobar que la aplicación se comporta exactamente como se describe en nuestros escenarios `*.feature`. Además, Behat puede automatizar funcionalidades relacionadas con web mediante la librería Mink [Min16].

9.3.2.3. Rellenar las funciones

Una vez se tienen las funciones de la clase `FeatureContext` definidas, solo queda introducir los objetos que se necesiten (responsabilidades) y volver a ejecutar el siguiente comando (esta vez para iniciar los tests funcionales):

```
:dummy$ bin/behat
```

El sistema devuelve una serie de errores entre los que se encuentra un "**`PHP Fatal Error: ...`**".
En este punto, se está listo para picar el código. Behat ayuda en esta fase dando las instrucciones a seguir (puesto que las pruebas ya han sido pensadas y diseñadas), y hay que seguirlas teniendo en

mente la estructura explicada dos apartados atrás (Inicializar Banco). Con cada corrección hay que volver a ejecutar el comando anterior para obtener la siguiente instrucción.

Behat tiene muchas opciones para ejecutarse, entre las que se encuentra la opción de acotar las funciones a ejecutar. De este modo se puede comprobar una funcionalidad concreta o incluso una de las funciones de una funcionalidad:

- Filtrar por funcionalidad:
  `:dummy$ bin/behat features/`**`nombreDeLaFuncionalidad`**`.feature`
- Filtrar por título de una funcionalidad:
  `:dummy$ bin/behat features/`**`nombreDeLaFuncionalidad`**`.feature --name "parte del título"`

En versiones anteriores de Behat [DB16] también se podía especificar el formato de salida de las pruebas (entre los que se encontraban HTML y JUnit), pero esa funcionalidad no ha sido desarrollada en la versión actual (3.0).

9.4. Herramienta de rastreo: Mink

Una de las partes más importantes de una web es el navegador, que es la ventana a través de la que los usuarios interactúan con la aplicación y con otros usuarios. Por lo tanto, los usuarios siempre hablan con la aplicación web a través del navegador. Esto significa que, para probar el correcto funcionamiento de nuestra aplicación web, se precisa simular de alguna forma la interacción entre el navegador y la aplicación.

Hay un gran número de emuladores de navegadores (Goutte [Gou16], Selenium [Sel16], Sahi [Sah16], etc.) que hacen el mismo trabajo pero de distinto modo. Los distintos emuladores se comportan de modo distinto y tienen diferentes API's (de inglés, Application Programming Interface), pero están divididos en dos grupos bien diferenciados:

- Emuladores de navegadores sin interfaz[20] gráfica
- Controladores de navegadores

Los emuladores sin interfaz gráfica son implementaciones de especificaciones en HTTP simple (como por ejemplo, Goutte). Estos envían una petición HTTP contra la aplicación y tratan el contenido de la respuesta. Son sencillos de configurar y ejecutar gracias a que pueden escribirse en cualquier lenguaje de programación y ejecutarse a través de una consola en servidores sin interfaz gráfica. Las ventajas de este tipo de emuladores son:

- la simplicidad,
- la velocidad y
- la habilidad de ejecutarse sin necesitar un navegador real.

Por contra, tienen el gran inconveniente de no soportar JS/AJAX.

Por otro lado, los controladores de navegadores pretenden, como su propio nombre indica, controlar el navegador (son programas que controlan a otros programas). Estos simulan la interacción del usuario con el navegador y tienen la capacidad de recuperar la información de la página del navegador en la que se encuentran (como por ejemplo, Selenium y Sahi). La gran ventaja de este tipo de emuladores es que soportan JS/AJAX.
Por contra, tienen los inconvenientes de:

[20] f. Conexión, física o lógica, entre una computadora y el usuario, un dispositivo periférico o un enlace de comunicaciones.

- requerir tener el navegador instalado,
- precisar cierta configuración adicional y
- ser mucho más lentos que los emuladores sin interfaz gráfica.

Por norma general se debe escoger el emulador que mejor se adapte al proyecto en el que va a utilizarse y usar su API para realizar las pruebas, pero, como se ha explicado con anterioridad, ambos tipos de emuladores tienen ventajas e inconvenientes. Si se utiliza un emuladores sin interfaz gráfica, no se podrán probar los JS/AJAX de las páginas. Y si se utiliza un controlador de navegador, llegará un punto en el que todo el banco de pruebas se volverá muy lento. Por esto es por lo que hay que usar ambos tipos de emuladores, y aquí es donde se necesita Mink.

Mink es un controlador/emulador open source para aplicaciones web, escrito en PHP 5.3. La peculiaridad de Mink es que elimina las diferencias entre las API's de los distintos emuladores ofreciendo distintos drivers (del inglés, controladores[21]) para cada emulador además de ofrecer un modo sencillo de controlar el navegador [MCB16], recorrer las páginas [MTP16], manipular elementos de las páginas [MMP16] o interactuar con las páginas [MIP16].

9.4.1. Instalación

Para instalar Mink hay que asegurarse de que se tiene al menos instalada la versión de PHP 5.3.1. Para ello hay que abrir el terminal y ejecutar el siguiente comando:

```
:dummy$ php -v
```

En nuestro caso, tenemos instalada la versión PHP 5.5.34.
A continuación se instala Mink con Composer ejecutando el siguiente comando:

```
:dummy$ composer require behat/mink
```

Esto lo instalará todo dentro de la carpeta *vendor*.

Hay que tener en cuenta que, por defecto, Mink se instala sin drivers. Los drivers son el modo que tiene Mink de ofrecer una API consistente para librerías de navegadores distintos y escritas, a veces, en lenguajes distintos. Estos drivers son clases simples que implementan una interfaz. La interfaz contiene los métodos puente entre Mink y los emuladores del navegador. Por lo tanto, Mink siempre se comunica con los emuladores de los navegadores a través de sus drivers.

Esto significa que, para usar cualquier driver adicional, hay que instalarlos con Composer requiriendo las dependencias apropiadas teniendo en cuenta las limitaciones e incompatibilidades de cada uno de ellos.

Feature	BrowserKit / Goutte	Selenium2	Zombie	Selenium	Sahi
Page traversing	Yes	Yes	Yes	Yes	Yes
Form manipulation	Yes	Yes	Yes	Yes	Yes
HTTP Basic auth	Yes	No	Yes	No	No
Windows management	No	Yes	No	Yes	Yes
iFrames management	No	Yes	No	Yes	No

[21] m. Programa que permite a una computadora manejar los componentes que tiene instalados.

Request headers access	Yes	No	Yes	No	No
Response headers	Yes	No	Yes	No	No
Cookie manipulation	Yes	Yes	Yes	Yes	Yes
Status code access	Yes	No	Yes	No	No
Mouse manipulation	No	Yes	Yes	Yes	Yes
Drag'n Drop	No	Yes	No	Yes	Yes
Keyboard actions	No	Yes	Yes	Yes	Yes
Element visibility	No	Yes	No	Yes	Yes
JS evaluation	No	Yes	Yes	Yes	Yes
Window resizing	No	Yes	No	No	No
Window maximizing	No	Yes	No	Yes	No

Tabla 1: Características de los drivers de Mink

9.4.1.1. Instalación Selenium2Driver

Cuando se instala Mink, por defecto viene configurado para ser usado con el Driver Goutte, por lo que para que nuestro proyecto pueda probar las páginas que contienen JS/AJAX además de para poder hacer capturas de pantalla de los navegadores automáticamente, se ha instalado Selenium2Driver [Sel2].

Selenium2Driver proporciona un puente para usar la herramienta web Selenium [Sel16] con Mink. Para poder usarlo hay que:

- Instalar Selenium2Driver con composer:
  ```
  :dummy$ composer require behat/mink-selenium2-driver
  ```
- Descargar el servidor de Selenium de la página:
  http://docs.seleniumhq.org/download/
- Abrir el terminal y ejecutar el siguiente comando (la versión ha de ser la descargada en el paso anterior):
  ```
  :dummy$ java -jar selenium-server-standalone-2.53.0.jar
  ```

En este punto ya se puede usar el driver de Selenium además del de Goutte y BrowserKit para probar los Casos de Uso del proyecto. Esto significa que se podrán llevar a cabo pruebas visuales (y sus correspondientes capturas de pantalla) en Controladores de Navegadores, función no disponible en Emuladores de Navegadores sin interfaz gráfica (Goutte, BrowserKit...).

9.4.1.1.1. Instalación ChromeDriver

Cuando se instala el Driver Selenium2, por defecto viene configurado para ser usado con el navegador Firefox [Fir], por lo que para que nuestro proyecto sea multi-navegador se ha posibilitado el uso con el navegador Chrome [Chr]. Para ello se ha instalado en el entorno de trabajo el ChromeDriver [ChD].

ChromeDriver es un ejecutable que el driver Selenium2 instalado (WebDriver [SWI]) usa para controlar Chrome. Lo mantiene el equipo de Chromium [Chu] con la ayuda de contribuyentes de WebDriver. Para poder usar ChromeDriver hay que:

- Asegurarse de tener el navegador Chromium o Google Chrome instalado en una localización reconocible. Esto significa que ChromeDriver espera encontrarlo en la localización por defecto de nuestro entorno, y de no ser así, habría que forzar que ChromeDriver usase una localización personalizada estableciendo una capacidad especial. En nuestro caso se tiene instalado en la localización por defecto.
- Descargar la última versión (en el momento de realización de este proyecto era la 2.22) del binario de ChromeDriver para nuestro SO de la página de descargas:
  https://sites.google.com/a/chromium.org/chromedriver/downloads
- Ayudar a que WebDriver halle el ejecutable ChromeDriver descargado incluyendo su localización en la variable de entorno de nuestro PATH.

En este punto, ya se puede usar el driver de Chrome además del de Firefox para probar los Casos de Uso del proyecto, por lo tanto se podrán llevar a cabo pruebas multi-navegador.

9.4.2. Controlar el Navegador

Antes que nada, hay que escoger el driver que se desea utilizar. Una vez escogido el driver, se va al punto de entrada del navegador. Ese punto de entrada se llama sesión, y representa el concepto de la ventana de un navegador. Puesto que como usuario lo primero que se hace es abrir una pestaña en el navegador, lo primero que hay que hacer después de escoger el driver es iniciar una sesión. En el caso de que se escoja el driver de Goutte, se iniciará del siguiente modo:

```
$driver = new \Behat\Mink\Driver\GoutteDriver();          # objeto driver
$session = new \Behat\Mink\Session($driver);              # representa el navegador
$session->start();                                        # representa una pestaña
```

El siguiente paso es decirle al navegador la página a visitar, por lo que se le ha de indicar a la sesión el hecho de visitar la página. En nuestro caso, puesto que se trata de un proyecto local, se hará del siguiente modo:

```
$session->visit('http://localhost:8000/');                # visitar la página de la aplicación
```

9.4.3. Recorrer una Página

Las pruebas que se llevan a cabo en una aplicación web suelen requerir recorrer sus páginas. En Mink esto se puede hacer gracias a su API *Element*, que permite recorrer las páginas, manipular sus elementos e interactuar con ellos.

La API *Element* contiene dos clases principales que representan el documento (`DocumentElement`) y sus elementos (`NodeElement`). La instancia del documento se accede del siguiente modo:

```
$page = $session->getPage();              # representa la página
```

9.4.4. Manipular una página

Una vez obtenida la página a probar, el siguiente paso es usar los métodos que ofrece Mink para manipularla y tratar sus elementos. Los métodos más comunes son los siguientes:

1. Obtener el nombre de un tag.
2. Acceder a los atributos HTML.
3. Obtener Contenido, Texto y Visibilidad de un elemento.

9.4.4.1. Obtener el nombre de un tag

Para obtener el nombre de la etiqueta de un elemento se usa el método `getTagName`, que devuelve los nombres en minúsculas. Su uso es como sigue:

```
$el = $page->find('css', '.something');
// obtener el nombre de la etiqueta
echo $el->getTagName(); // displays 'a'
```

9.4.4.2. Acceder a los atributos HTML

Para obtener los atributos HTML del elemento se tienen los siguientes métodos:

- **`NodeElement:hasAtribute`**
  Comprueba si se le ha dado un atributo al elemento.
- **`NodeElement:getAtribute`**
  Obtiene el valor del atributo del elemento.
- **`NodeElement:hasClass`**
  Comprueba si se le ha dado una clase al elemento.

Su uso es como sigue:

```
$el = $page->find('css', '.something');
if ($el->hasAttribute('href')) {
    echo $el->getAttribute('href');
} else {
    echo 'This anchor is not a link. It does not have an href.';
}
```

9.4.4.3. Obtener Contenido, Texto y Visibilidad de un elemento

Para obtener el contenido de los elementos se tienen los siguientes métodos:

- **`Element:getHtml`**
  Obtiene el HTML interno del elemento (es decir, todos los hijos del elemento).
- **`Element:getOuterHtml`**
  Obtiene el HTML externo del elemento (es decir, incluyendo al mismo elemento).
- **`Element:getText`**
  Obtiene el texto del elemento.
- **`NodeElement:isVisible`**

Comprueba la visibilidad del elemento.

9.4.5. Interactuar con una página

Las mayoría de pruebas que se llevan a cabo en una aplicación web suelen requerir interactuar con sus páginas abiertas en un navegador. En Mink esto se vuelve a hacer gracias a su API *Element*, que permite interactuar con los elementos de la página.

Las interacciones más típicas son las siguientes:

1. Links y Botones.
2. Formularios.
3. Ratón.
4. Teclado.

9.4.5.1. Links y Botones

Para interactuar con links y botones se tienen los siguientes métodos (internamente equivalentes):

- **`NodeElement:click`**
  Clica sobre el link de la página.
- **`NodeElement:press`**
  Presiona sobre el botón de la página.

9.4.5.2. Formularios

Para interactuar con formularios se tienen los siguientes métodos:

- **`NodeElement:getValue`**
  Obtiene el valor de un campo determinado del formulario.
- **`NodeElement:setValue`**
  Introduce el valor de un campo determinado del formulario.
- **`NodeElement:isChecked`**
  Comprueba si una casilla u opción está seleccionada.
- **`NodeElement:isSelected`**
  Comprueba si la opción `<option>` está seleccionada.
- **`NodeElement:check`**
  Selecciona una casilla.
- **`NodeElement:uncheck`**
  Deselecciona una casilla.
- **`NodeElement:selectOption`**
  Selecciona una opción concreta de un grupo de casillas u opciones.
- **`NodeElement:attachFile`**
  Adjunta un fichero en una entrada de ficheros.
- **`NodeElement:submit`**
  Envía el formulario.

9.4.5.3. Ratón

Para interactuar con formularios se tienen los siguientes métodos:

- **`NodeElement:click`**
  Hace un clic sobre el elemento.
- **`NodeElement:doubleClick`**

Hace un doble-clic sobre el elemento.

- **`NodeElement:rightClick`**
  Hace un clic derecho sobre el elemento.
- **`NodeElement:mouseOver`**
  Mueve el ratón sobre el elemento.

9.4.5.4. Teclado

Para interactuar con el teclado se tienen los siguientes métodos:

- **`NodeElement:keyDown`**
  Baja una tecla.
- **`NodeElement:keyPress`**
  Presiona una tecla.
- **`NodeElement:keyUp`**
  Suelta una tecla.

9.5. Herramienta de desarrollo web: Symfony 2

Los motivos por los que se decidió realizar el portal *dummy* con el marco de trabajo Symfony fueron su reputación, sus referencias y su recomendación.

Su reputación ha sido forjada gracias a la rápida adopción por parte de los profesionales en activo desde sus inicios en 2005 [SGR16], que contribuyeron a que a día de hoy Symfony sea estable además de internacionalmente conocido y reconocido. Es precisamente gracias a su madurez, por lo que la comunidad que le da soporte es muy extensa. Por supuesto también se dispone de las comunidades de software genéricas tipo stackoverflow [Sta16].

Sus referencias vienen dadas por la familia de proyectos que usan Symfony (algún componente o el marco de trabajo entero), ya sean privados u open-source. Solamente con mencionar algunos de ellos se ve la confianza de la comunidad software en este marco: Drupal, Joomla, Composer, Magento, Goutte, Doctrine, Behat, etc [PUS16].

La recomendación de usar Symfony vino dada por los componentes del equipo del proyecto, dada su experiencia anterior en proyectos que lo usaban e incluso en el desarrollo de aplicaciones web con Symfony.

9.5.1. Instalación y configuración

Puesto que al principio del proyecto, se trabajó con el sistema operativo (SO en adelante) Windows pero más adelante se decidió cambiar a OSX, a continuación se explican los pasos seguidos para instalar e iniciar la aplicación de Symfony con estos SO.
Symfony puede instalarse de distintas maneras [ICS16]:

- Con el instalador: método recomendado para crear nuevas aplicaciones Symfony.
- Sin el instalador: Si se usa PHP 5.3 o si no se puede ejecutar el instalador por algún motivo.

9.5.1.1. Con el instalador: OSX

Lo primero es crear un comando global de Symfony en el sistema. Para ello hay que abrir el terminal y ejecutar los siguientes comandos:

```
$ sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

9.5.1.1.1. Crear la aplicación

Para crear la aplicación (dummy) hay que ejecutar el siguiente comando:

```
$ symfony new dummy
```

9.5.1.2. Con el instalador: Windows

Lo primero es descargar el instalador. Para ello hay que abrir el terminal y ejecutar el siguiente comando:

```
c:\> php -r "readfile('https://symfony.com/installer');" > symfony
```

A continuación hay que mover el archivo "symfony" descargado al directorio de nuestro proyecto (projects):

```
c:\> move symfony c:\projects
```

Y ejecutar el siguiente comando:

```
c:\projects\> php symfony
```

9.5.1.2.1. Crear la aplicación

Para crear la aplicación (dummy) hay que ejecutar los siguientes comandos:

```
c:\> cd projects
c:\projects> php symfony new dummy
```

Estos comandos crean un nuevo directorio llamado "dummy" que contiene un proyecto nuevo basado en la versión más estable disponible de Symfony (en nuestro caso, la 3.0). Además, el instalador comprueba si el sistema cumple con los requisitos mínimos para ejecutar la aplicación Symfony. De no ser así, se verá una lista de cambios necesarios para cumplir con dichos requisitos.

Si se desea crear un proyecto con una versión concreta de Symfony, se debe usar el segundo argumento del comando "new":

```
#para usar la versión más reciente de una rama de Symfony
$ symfony new dummy 2.8

#para usar una versión de Symfony específica
$ symfony new dummy 2.7.3

#para usar una versión beta o RC (útil para probar nuevas versiones [TRP16] de Symfony)
$ symfony new dummy 3.0.0-BETA1
$ symfony new dummy 2.7.0-RC1
```

9.5.1.3. Sin el instalador: OSX

Éste método alternativo de instalación usa Composer [Com16], un gestor de dependencias que usan las aplicaciones PHP modernas.

1. Lo primero es instalar Composer de forma global. Para ello hay que abrir el terminal y ejecutar los siguientes comandos, que nos indicarán si alguna de las propiedades de nuestro archivo "php.ini" son incorrectas para después descargar la última versión de "composer.phar" en nuestro directorio:

```
        #descargar el instalador en el directorio actual
$ php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
        #verificar el SHA-384 del instalador
$ php -r "if (hash_file('SHA384', 'composer-setup.php') ===
'92102166af5abdb03f49ce52a40591073a7b859a86e8ff13338cf7db58a19f7844fbc0bb79b2773bf30791e935d
bd938') { echo 'Installer verified'; } else { echo 'Installer corrupt';
unlink('composer-setup.php'); } echo PHP_EOL;"
        #correr el instalador
$ php composer-setup.php
        #eliminar el instalador
$ php -r "unlink('composer-setup.php');"
```

Puesto que el SHA-384 del instalador es una verificación, variará con cada versión, por lo que hay que acceder al siguiente link para insertar el comando correcto:

https://getcomposer.org/download/

2. A continuación hay que mover el archivo "composer.phar" a un directorio que esté en el PATH del sistema ejecutando:

   $ sudo mv composer.phar /usr/local/bin/composer

9.5.1.4. Sin el instalador: Windows

Hay que descargar el instalador de https://getcomposer.org/Composer-Setup.exe, ejecutarlo y seguir las instrucciones.

9.5.1.5. Crear la aplicación Symfony con Composer

Para crear una aplicación Symfony (llamada dummy) basada en su última versión con Composer, hay que ejecutar el siguiente comando:

```
$ composer -vvv create-project symfony/framework-standard-edition dummy
```

Si se necesita una versión concreta de Symfony, hay que usar el segundo argumento del comando "create-project":

```
$ composer -vvv create-project symfony/framework-standard-edition dummy "3.0.*"
```

9.5.2. Correr la aplicación Symfony

Symfony aprovecha el proveedor de servicios interno de PHP para correr aplicaciones mientras se desarrollan. Por ello, para correr una aplicación Symfony simplemente hay que ir al directorio en el que se encuentra nuestro proyecto y ejecutar un comando que arranque el servidor:

```
$ cd dummy
:dummy$ php bin/console run:server
```

A continuación hay que abrir un navegador e introducir la siguiente URL:

http://localhost:8000/

Entonces, se debería visualizar la siguiente página de bienvenida:

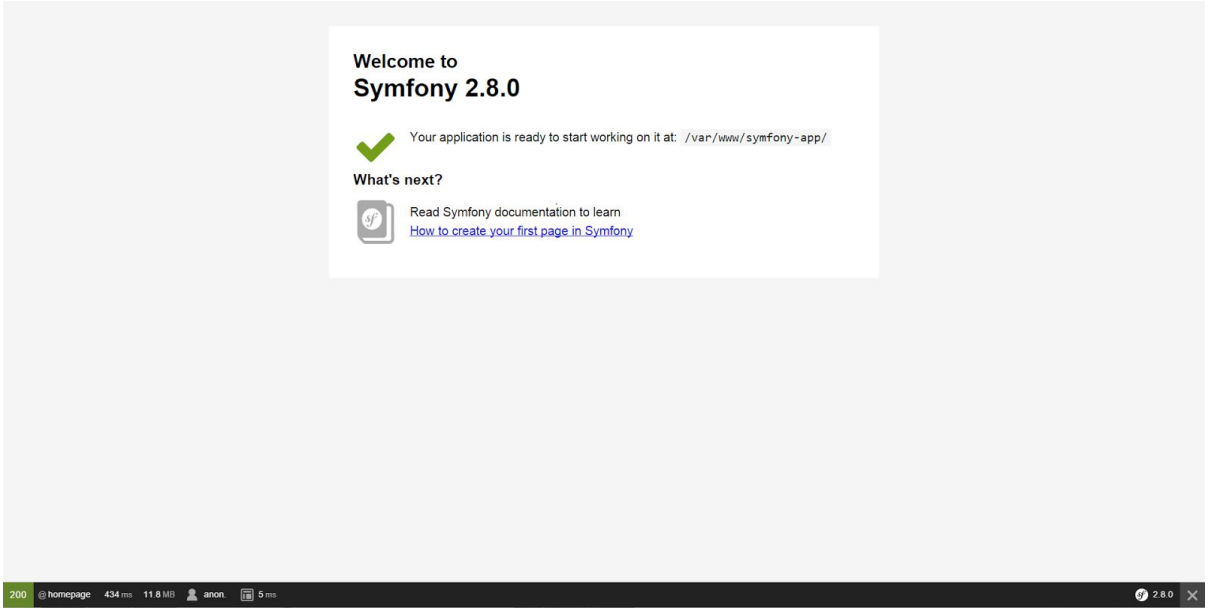


Imagen 2: Página de bienvenida

9.5.3. Comprobar configuración y Preparación

Para comprobar que nuestro entorno de trabajo está preparado para usar Symfony, la aplicación viene con un tester visual de configuración del servidor.
Por lo tanto, antes de empezar a desarrollar la aplicación, hay que llevar a cabo los siguientes pasos:

1. Correr la aplicación (apartado anterior a éste).
2. Abrir el navegador y acceder a la siguiente URL:
   http://localhost:8000/config.php
   Aquí se indicará si el entorno está preparado para usar Symfony. De haber alguna incidencia, deberá corregirse antes de continuar. Las recomendaciones no son obligatorias, pero suelen tratar de temas de eficiencia a tener en cuenta. En nuestro caso, la página recomienda instalar y activar la extensión "intl" (usada para los validadores) y un acelerador PHP.
   a. Instalar extensión "intl":

```
:dummy$ composer require symfony/intl
```

   b. Activar extensión "intl":
      i. Editar el fichero php.ini de XAMPP:

```
:etc$ vim php.ini
#descomentar la línea: extension=php_intl.dll
```

      ii. Reiniciar el servidor:

```
:dummy$ app/console server:stop
:dummy$ app/console server:run
```

   En cuanto al acelerador PHP, se consultó con el equipo si era imperioso instalarlo para optimizar la aplicación, y comunicaron que no era necesario, por lo que se decidió declinar esta recomendación.

9.5.4. Actualizar aplicación Symfony

En este punto ya se tiene una aplicación Symfony totalmente funcional en la que se puede desarrollar el proyecto. Cada aplicación Symfony depende de una serie de librerías externas. Éstas se descargan en el directorio "vendor/" y son gestionadas por Composer.
Se recomienda actualizar las librerías externas con frecuencia para prevenir bugs y vulnerabilidades de seguridad. Para actualizarlas todas a la vez hay que ejecutar el siguiente comando:

```
:dummy$ composer update
```

Para comprobar si alguna dependencia tiene vulnerabilidades, se ejecuta el siguiente comando:

```
:dummy$ app/console security:check
```

Una vez instalado y actualizado todo, se pasa a crear las primeras páginas del proyecto [CFP16].

9.5.5. Estructura de ficheros

En la primera fase del desarrollo del proyecto se usó la última versión de Symfony (3.0). La estructura de ficheros de esta versión es la siguiente:

```
/app    # los ficheros de configuración principal y la capa de HTML básica
        /config
        /Resources
                /views
/src    # nuestro código (PHP) agrupado en Bundles
        /AppBundle
        /Controller
/vendor # librerías externas (bajadas por Composer)
/web    # único directorio accesible públicamente: controladores frontales (app_dev.php, app.php)
        /bundles
/bin    # contiene la consola (generadores, cache, assets)
```

Sin embargo, a medida que se avanzó en el desarrollo del proyecto, se vio que había vendors incompatibles con la última versión, por lo que se decidió bajarla a una mucho más usada (2.8). La estructura de ficheros de esta versión es la siguiente:

```
/app    # los ficheros de configuración principal y la capa de HTML básica
        /console
        /cache
        /config
        /logs
        /Resources
                /views
/src    # nuestro código (PHP) agrupado en Bundles
        /AppBundle
                /Controller
/vendor # librerías externas (bajadas por Composer)
/web    # único directorio accesible públicamente: controladores frontales
        /app.php
        /app_dev.php
```

9.5.6. Controlador

En symfony, un controlador crea y devuelve un objeto *response* al cliente. El proceso hasta entonces es el siguiente:

1. Un *front controller* (`app.php`, `app-dev.php`):
   - maneja las peticiones,
   - e inicia la aplicación.
2. El *router* (`routing.yml`, `routing_dev.yml`):
   - lee información de la petición,
   - encuentra una ruta que coincida con la información,
   - y lee el parámetro "_controller" de la ruta.
3. El controlador de la ruta hallada:
   - es ejecutado,
   - el código del controlador crea y devuelve un objeto *response.*
4. Se devuelven al cliente:
   - las cabeceras,
   - el contenido HTTP del objeto *response.*

9.5.7. Plantillas TWIG

Una plantilla es un archivo de texto que puede generar cualquier otro formato basado en texto (HTML, XML, CVS, LaTex, etc.) Las plantillas PHP son las más populares, sin embargo Symfony2 incluye un lenguaje de plantillas llamado Twig que es mucho más potente y elegante que PHP [SLO16].

La sintaxis de Twig se basa en un conjunto de etiquetas especiales [STE]:

- `{{ ... }}`: Decir algo.
  - imprimir una variable,
  - imprimir el resultado de una expresión.
- `{% ... %}`: Hacer algo, controla la lógica de la plantilla.
  - ejecutar declaraciones (loops tipo *for*).
- `{# ... #}`: Comentar algo, añadir comentarios de línea o párrafo

Además, Twig dispone de:

- FILTROS: Modificar contenido antes de ser renderizado.
  - `{{ title | upper }}` Convierte la variable "title" a mayúsculas.
- TAGS: Hacen alguna acción en el control de la plantilla.
  - `{% for i in 0..10 %}` Hace un bucle desde `i=0` hasta `i=10`

```
        <div class = "{{ cycle( ['odd', 'even'], i ) }}">
                <!-- código HTML -->
        </div>
{% endfor %}
```

9.5.8. Instalar un Bundle

Tal y como se define en *Symfony 2.4, el libro oficial* [SLO16], un *bundle* es un concepto parecido al que en otras aplicaciones denominan *plugin*, la diferencia es que en Symfony2 todo son bundles, desde las funcionalidades básicas del entorno hasta nuestro código de la aplicación. Los bundles permiten utilizar funcionalidades construidas por terceros o empaquetar nuestras funcionalidades para distribuirlas y reutilizarlas en otros proyectos. Por lo tanto, un bundle es un conjunto de

archivos estructurados de una forma concreta, en un directorio concreto y que implementa una característica concreta. Puesto que en Symfony todo son bundles, viene con un conjunto de bundles por defecto, como el de *Doctrine* [DOC]. Este bundle es una librería necesaria para la persistencia y lectura de la información de y a una BBDD.

El uso de bundles de terceros facilita el desarrollo de aplicaciones web, puesto que evita tener que "reinventar la rueda". Un ejemplo de bundle (muy conocido) sería el *FOSUserBundle* [FOS]. Este bundle añade la funcionalidad de tener un sistema de usuarios conectados con la BBDD, por lo que los usuarios pueden almacenarse vía ORM en Doctrine, vía ODM en MongoDB y en CouchDB o vía Propel. También soporta registros (con opción de confirmación por mail), reseteo de contraseñas, etc.

Los pasos a seguir para instalar un bundle son los siguientes:

1. Buscar el bundle en http://www.knpbundles.com.
2. Buscar en https://packagist.org las dependencias del bundle e introducirlas en el fichero ***composer.json***.
3. Actualizar composer:
   ```
   :dummy$ composer update
   ```
4. Configurar el Bundle (como diga su creador) a través del archivo:
   `app/config/`**`config.yml`**
5. Activar el Bundle (líneas de instanciación) editando el archivo:
   `app/`**`AppKernel.php`**

Sin embargo, hay algunos casos en los que puede convenir no hacer uso de bundles de terceros, como lo es el nuestro. Ésto es así porque los bundles instalan muchas cosas para poder funcionar aisladamente que, en nuestro caso, complicarían el portal. Para mantenerlo sencillo y ágil, nos sale a cuenta implementar nosotros mismos la funcionalidad de *login* que nos proporcionaría el bundle.

9.5.9. Crear una BBDD

Para crear la BBDD del sistema hay que seguir los siguientes pasos:

1. Configurar la información de conexión de la BBDD en el archivo:
   `app/config/`**`parameters.yml`**
   De este fichero solamente hace falta editar las siguientes líneas:
   ```
   database_host: host_de_nuestro_portal
   database_port: 'puerto_de_nuestra_BBDD'
   database_name: nombre_de_nuestra_BBDD
   database_user: usuario_de_nuestra_BBDD
   database_password: 'contraseña_de_nuestro_usuario'
   ```
2. Crear el esquema UML.
3. Crear el contenedor de las tablas del esquema anterior en phpmyadmin (activar XAMPP).
4. Crear los ficheros ".orm" dentro de `/src`.
5. Crear las tablas de la BBDD y sus relaciones con el comando:
   ```
   :dummy$ app/console doctrine:database:create
   ```

9.5.10. Crear una página nueva

Para crear una página en symfony hay que seguir los siguientes pasos:

1. Crear una ruta (URL) en el fichero:
   `app/config/`**`routing.yml`**
   Si la ruta fuese la de la página de *login*, se crearía como sigue:

```
login:
   path:       /login
   defaults:   { _controller: AppBundle:Login:index }
```

2. Crear su controlador (función que construye la página). Si fuese la página de *login*, sería el fichero:

   src/AppBundle/Controller/**LoginController.php**

   Y su contenido inicial sería el siguiente:

```
<?php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;

class LoginController extends Controller
{
   /**
    * /login :: indexAction
    */
   public function indexAction(Request $request)
   {
       return $this->render('AppBundle:login:login.html.twig');
   }
}
```

10. Especificación (testing)

Los tests que se decidieron llevar a cabo sobre el portal *dummy* son los requisitos funcionales, es decir: navegación, entrada de datos, procesamiento y obtención de resultados. Por ello se generó un fichero de Excel con las pruebas que consideramos más importantes. Posteriormente, este fichero fué estudiado con el team-leader del equipo y sus funcionalidades fueron ordenadas por relevancia.

Las pruebas llevadas a cabo están divididas por páginas, y dentro de cada página, por acciones. Así pues, los tests han sido los siguientes:

1. LOGIN
2. CREAR NUEVO USUARIO
3. NAVEGACIÓN
    a. MENÚ
    b. POST
    c. ABOUT US

10.1. Login

El mockup de la página de *login* era el siguiente:

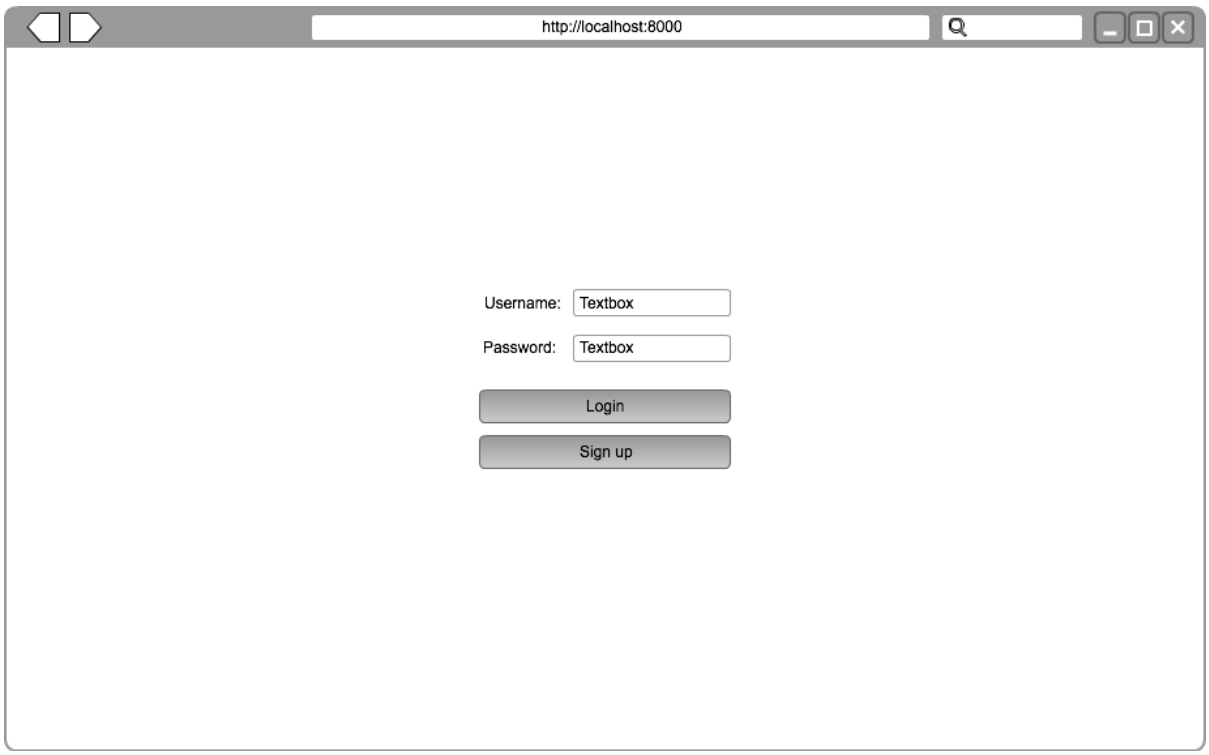


Imagen 8: Mockup de la página de login

Por lo tanto, los casos considerados para testear la página fueron los siguientes:

Username:

- Existe en la BBDD
- No existe en la BBDD. Debe mostrar un mensaje de error.

El mensaje de error se debía mostrar como en el mockup que sigue:

Imagen 9: Mockup de la página de login con error

10.2. Crear nuevo usuario

El mockup de la página de *sign up* era el siguiente:

Imagen 10: Mockup de la página de sign up

Por lo tanto, los casos considerados para testear la página fueron los siguientes:

Mail:

- Válido (contiene una @, un dominio...).
- Ha sido confirmado (link al correo).
- Único en la BBDD.
- No válido. Debe mostrar un mensaje de error.
- No confirmado. Debe eliminar el usuario (al cabo de X días).
- No único en la BBDD. Debe mostrar un mensaje de error.

Username:

- Mayor de 5 caracteres.
- Contiene al menos un número.
- Contiene al menos un símbolo ($,%,&,*,^...).
- Distinto del username.
- Contenido en UTF8-bin.
- Menor o igual a 5 caracteres. Debe mostrar un mensaje de error.
- No contiene al menos un número. Debe mostrar un mensaje de error.
- No contiene al menos un símbolo ($,%,&,*,^...). Debe mostrar un mensaje de error.
- Igual al username. Debe mostrar un mensaje de error.

Password:

- Único en la BBDD.

- Mayor de 5 caracteres.
- Contenido en UTF8-bin.
- No único en la BBDD. Debe mostrar un mensaje de error.
- Menor o igual a 5 caracteres. Debe mostrar un mensaje de error.

Los mensajes de error se debían mostrar como en el mockup que sigue:

Imagen 11: Mockup de la página de sign up con error

10.3. Navegación

10.3.1. Menú

La navegación por el portal web se debería hacer a través de las pestañas, por lo que los casos considerados para testear la navegación fueron los siguientes:

Home:

- Carga la página.

About us:

- Carga la página.

Exit:

- Se cierra la sesión y se carga la página de *login*.

10.3.2. Home

El mockup de la página de *home* era el siguiente:

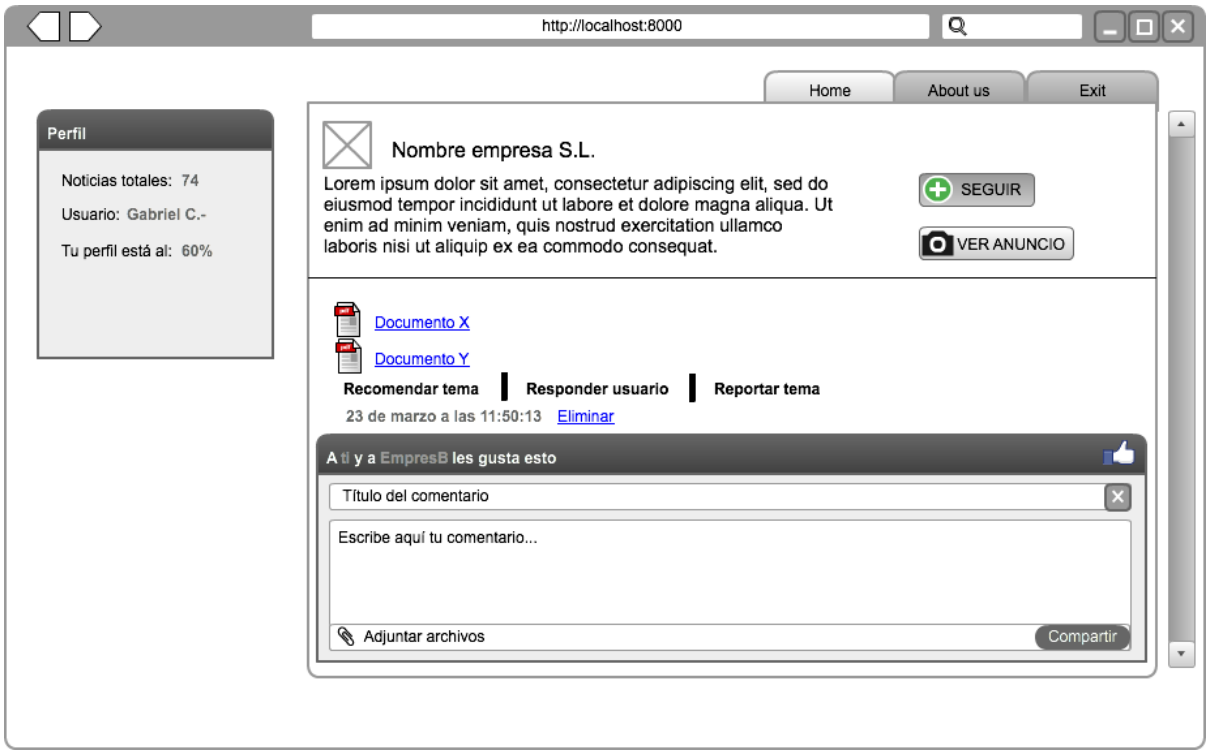


Imagen 12: Mockup de la página de home

Por lo tanto, los casos considerados para testear la página fueron los siguientes:

Seguir empresa:

- Clicar en "SEGUIR". El usuario debe seguir a la empresa.

Dejar de seguir empresa:

- Clicar en "NO SEGUIR". El usuario deja de seguir a la empresa.

Gustar un post:

- Clicar en icono de "like". Al usuario le gusta el anuncio.

Dejar de gustar un post:

- Clicar en icono de "dislike". Al usuario deja de gustarle el anuncio.

Comentar anuncio:

- SIN ADJUNTOS: Escribir un comentario y clicar en "COMPARTIR". Se comparte el comentario del usuario.
- CON ADJUNTOS: Escribir un comentario, adjuntar archivos y clicar en "COMPARTIR". Se comparte el comentario del usuario con los adjuntos.

Eliminar comentario anuncio:

- Clicar en icono de "borrar". El usuario borra su comentario.

Ver anuncio:

- Clicar en "VER ANUNCIO". Se visualiza el anuncio.

10.3.3. About us

El mockup de la página de *home* era el siguiente:

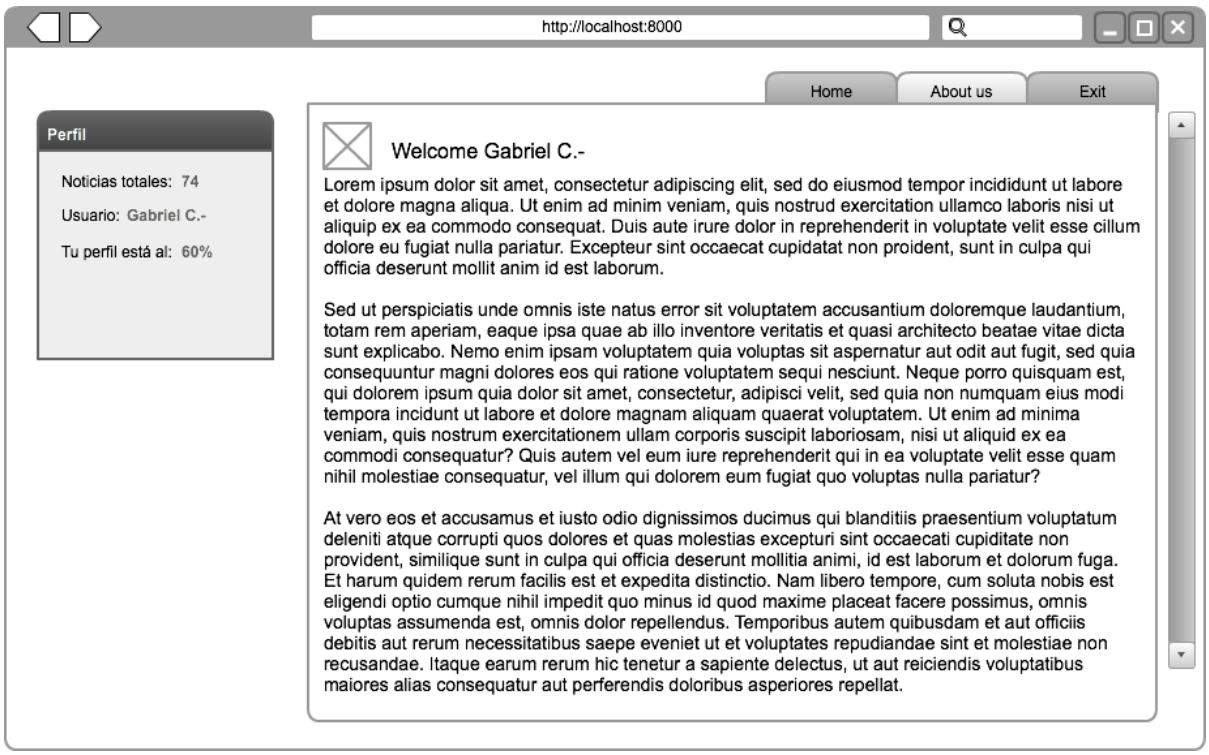


Imagen 13: Mockup de la página de about us

Por lo tanto, puesto que se trata de una página estática, el caso considerado para testear la página fue el de navegación por el menú y su carga (contemplado en un apartado anterior).

Así pues, se obtuvo el siguiente fichero de excel en el que se especificaron todas las pruebas a llevar a cabo:

		IdCaso	Estado	Caso	Precondiciones	Resultado ESPERADO
LOGIN						
	Username					
		1	X	Existe en la BBDD		
				Contenido en UTF8-bin		
		2	X	No existe en la BBDD		Mensaje de error
	Password					
				Mayor de 5 caracteres		
				Contiene al menos un número		
				Contiene al menos un símbolo ($,%,&,*,^...)		
				Contenido en UTF8-bin		
				Corresponde al username		
				Menor o igual a 5 caracteres		Mensaje de error
				No contiene al menos un número		Mensaje de error
				No contiene al menos un símbolo ($,%,&,*,^...)		Mensaje de error
				No corresponde al username		Mensaje de error
CREAR NUEVO USUARIO						
	Mail					
		3	X	Válido (contiene una "@")		
				Confirmado (link al mail)		
				Único en la BBDD		
		4	X	No válido (no contiene una "@")		Mensaje de error
				No confirmado (link al mail)		Eliminar usuario (a los X días)
				No único en la BBDD		Mensaje de error
	Username					
				Único en la BBDD		
		3	X	Mayor de 5 caracteres		
				Contenido en UTF8-bin		
				No único en la BBDD		Mensaje de error
		4	X	Menor o igual a 5 caracteres		Mensaje de error
	Password					
		3	X	Mayor de 5 caracteres		
				Contiene al menos un número		
				Contiene al menos un símbolo ($,%,&,*,^...)		
				Distinto del username		
				Contenido en UTF8-bin		
		4	X	Menor o igual a 5 caracteres		Mensaje de error
				No contiene al menos un número		Mensaje de error
				No contiene al menos un símbolo ($,%,&,*,^...)		Mensaje de error
				Igual al username		Mensaje de error
	Rol					
				Alguno de los disponibles en la BBDD		
	Enterprise_id					
				Alguno de los disponibles en la BBDD		
NAVEGACIÓN MENÚ					usuario logeado	
	Inicio					
		5	X	Carga la página		Se carga la página de "Inicio"
	Anuncios					
		6	X	Carga la página		Se carga la página de "Anuncios"
	About us					
		7	X	Carga la página		Se carga la página de "About us"
NAVEGACIÓN POST					usuario logeado	
	Seguir empresa					
		8	not implemented	Clicar en "SEGUIR"		El usuario sigue a la empresa
	Dejar de seguir empresa					
		9	not implemented	Clicar en "NO SEGUIR"		El usuario deja de seguir a la empresa
	Gustar un post					
		10 OPCIONA	not implemented	Clicar en icono de "like"		Al usuario le gusta el anuncio
	Dejar de gustar un post					
		11 OPCIONA	not implemented	Clicar en icono de "dislike"		Al usuario deja de gustarle el anuncio
	Comentar anuncio					
	SIN ADJUNTOS	12	X	Escribir un comentario y clicar en "COMPARTIR"		Se comparte el comentario del usuario
	CON ADJUNTOS		not implemented	Escribir un comentario, adjuntar archivos y clicar en "COMPARTIR"		Se comparte el comentario del usuario con los adjuntos
	Eliminar comentario anuncio					
			X	Clicar en botón de "borrar"		Al usuario deja de gustarle el anuncio
	Ver anuncio					
		13	not implemented	Clicar en "VER ANUNCIO"		Se visualiza el anuncio
NAVEGACIÓN ABOUT US					usuario logeado	

Imagen 14: Pruebas llevadas a cabo en el portal dummy

Como puede comprobarse, hay funcionalidades no implementadas en el portal. Además, se estudiaron las pruebas con el *team-leader* del equipo y se llegó a la conclusión de que las pruebas importantes a implementar serían las que se numeraron del 1 al 13, donde 1 es la más importante y 13 la menos importante. A medida que se fué desarrollando el portal, se fue descubriendo la necesidad de más pruebas (especificadas todas más adelante).

Por lo tanto, en los tests llevados a cabo hay 3 partes bien diferenciadas:

1. **Features**: son las funcionalidades que van a probarse. Una funcionalidad se testea probando sus distintos escenarios. Por lo tanto, un escenario es un caso específico que prueba parte de una funcionalidad. En nuestro caso concreto, una *feature* sería la página de *sign up*, y sus *scenarios* serían probar que el mail, username, password, rol y empresa se hayan introducido correctamente.
2. **Objeto de prueba**: es la página o portal sobre el que se van a llevar a cabo los tests. En nuestro caso concreto, se trataría del portal dummy.
3. **Resultados**: son los índices que marcan si una prueba ha pasado con éxito o no. En nuestro caso concreto sería la respuesta de Behat que obtendríamos por terminal y las capturas de pantalla cuando se usase el servidor de Selenium (necesario cuando la página a testear contiene AJAX/JS). Las capturas de pantalla se guardan automáticamente en las carpetas oportunas, y la respuesta del terminal puede redirigirse al mismo lugar como ficheros *.txt para que sean persistentes.

El esquema de los tests sería el siguiente:

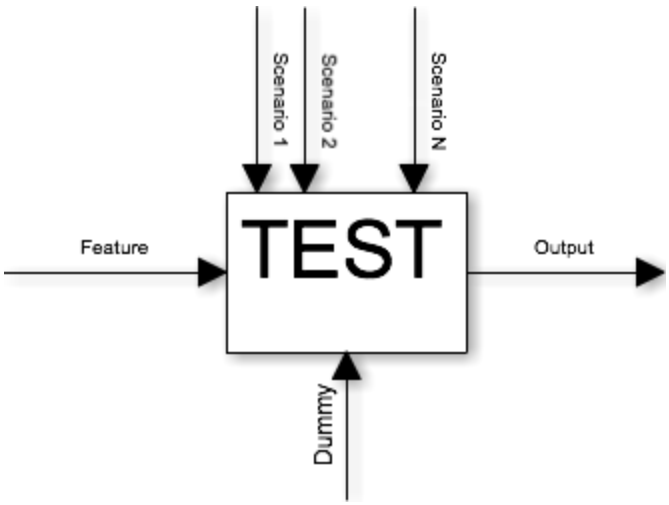


Imagen 15: Especificación de los tests

11. Diseño

El diseño del proyecto consta de distintas partes:

1. Arquitectura del Contexto
2. Arquitectura de las pruebas

11.1. Arquitectura Contexto

La arquitectura del Contexto del proyecto es la siguiente:

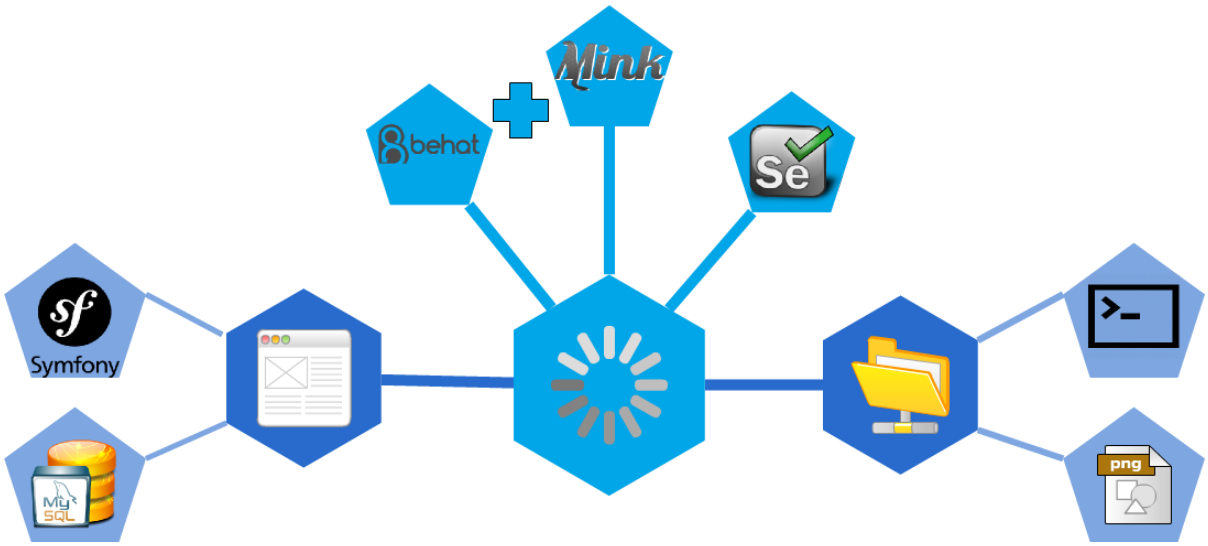


Imagen 16: Arquitectura Contexto

Como puede observarse en la imagen, hay tres partes bien diferenciadas. La de la izquierda, la del centro y la de la derecha, correspondiendo respectivamente el objeto de prueba desarrollado (portal dummy), la aplicación de testeo desarrollada y los resultados obtenidos.

11.1.1. Portal dummy

La arquitectura del portal *dummy* del proyecto es la siguiente:

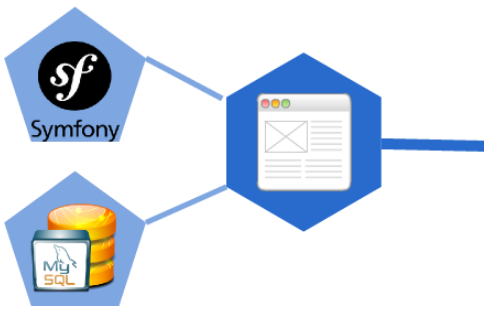


Imagen 17: Arquitectura Contexto: portal dummy

Como ya se explicó en apartados anteriores, el portal fue desarrollado en Symfony, accediendo a una BBDD MySQL. Este es el portal sobre el que se harán todas las pruebas.

11.1.2. Aplicación de testeo

La arquitectura de la aplicación de testeo del proyecto es la siguiente:

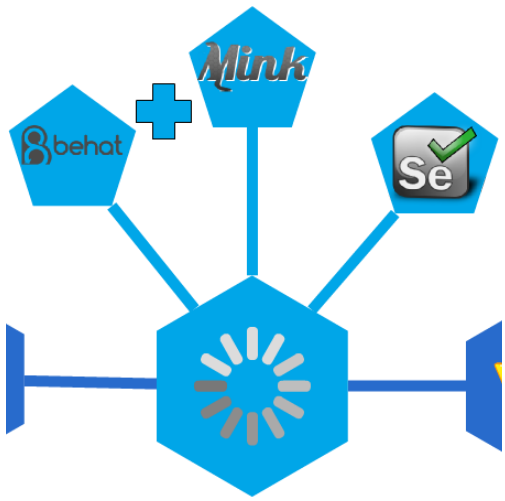


Imagen 18: Arquitectura Contexto: aplicación de testeo

Para llevar a cabo las pruebas, se han utilizado las siguientes partes:

- Behat: para definir las pruebas entre desarrolladores y *stakeholders* en BDD
- Mink: para rastrear dentro de la página en busca de las partes a testear
- El servidor Selenium: para las siguientes situaciones:
  - Cuando se quiere obtener capturas de pantalla de las pruebas realizadas (se guardarán automáticamente donde corresponda).
  - Cuando la página a testear contenga AJAX/JS, en cuyo caso es obligatorio el uso del servidor Selenium.

Como ya se explicó en apartados anteriores, Behat utiliza el lenguaje Cucumber por debajo. Esto permite poder describir las funcionalidades en un lenguaje que tanto desarrolladores como *stakeholders* comprenden. Por lo tanto, Behat es la rama de PHP de Cucumber, al igual que SpecFlow [Spe] es la rama de .NET. Estos dos lenguajes de programación tienen su propia rama dentro de Cucumber debido a su importancia, pero otros muchos lenguajes de programación (JS, Python, Ruby, Java...) pueden hacer uso de Cucumber.

Por lo tanto, la arquitectura del proyecto junto con las distintas ramas de Cucumber sería como se detalla en la siguiente imagen:

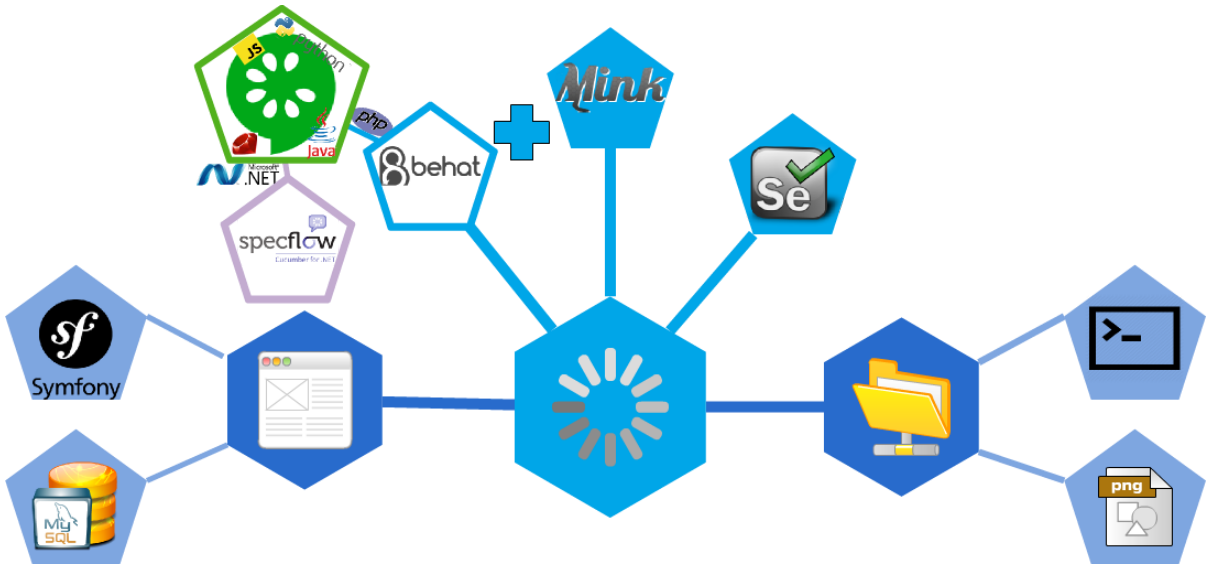


Imagen 19: Arquitectura Contexto con ramas de Behat

Para poder llevar a cabo las pruebas, hay que editar el fichero de properties de la aplicación especificando los Casos de Uso que se quieren ejecutar y el Driver con el que se desea probar. En nuestro caso concreto se tienen los siguientes:

Drivers:

1. Goutte.
2. BrowserKit.
3. Selenium2 (Firefox).
4. Selenium2 (Chrome).

Casos de Uso:

a. Login
b. Sign up
c. Menu
d. Comment (*)

(*) En el caso de querer probar el Caso de Uso *Comment* (caso **d**), es necesario usar el *Driver* Selenium2 con Chrome (caso **4**), puesto que la página a probar usa JS/AJAX.

Hay que destacar que, siempre que no sea imprescindible tener capturas de pantallas de los resultados ni se vaya a probar la página de *Comment* (usa JS/AJAX), es recomendable usar el *Driver* Goutte o BrowserKit. Esto es así por dos motivos:

- El *Driver* Selenium2 precisa levantar antes su servidor
- Los navegadores *headless*, son mucho más rápidos al ejecutarse.

11.1.3. Resultados obtenidos

La arquitectura de los resultados obtenidos del proyecto es la siguiente:

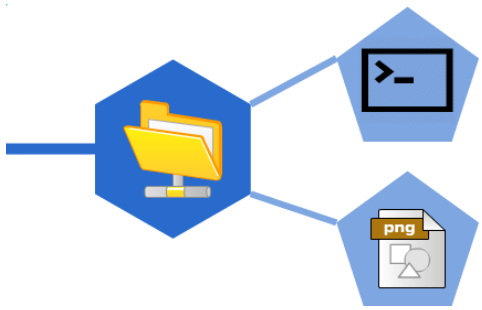


Imagen 20: Arquitectura Contexto: resultados obtenidos

Los resultados obtenidos en las pruebas realizadas son, por defecto, a través del terminal. Esto es así porque el uso de BDD es para llevar a cabo un desarrollo orientado a calidad, pero precisamente gracias a obtenerse por terminal, puede redirigirse la salida en un fichero `*.txt` en la ubicación que se considere (debería ser la de la prueba llevadas a cabo).

11.2. Arquitectura tester

Puesto que Behat está pensado para aplicar BDD, los ficheros en los que se especifican las Features a probar están dentro del proyecto del portal a probar. La idea es desarrollar ese portal gracias a Behat. Por lo tanto, en el proyecto *dummy* hay una carpeta *features* que contiene todas las funcionalidades y sus clases.

El formato de las llamadas de Behat puede consultarse abriendo el terminal y ejecutar el siguiente comando:

```
:dummy$ bin/behat -h
```

De este modo se observan todos los formatos que proporciona Behat para hacer las llamadas. La información obtenida es la siguiente:

```
Usage:
  behat [options] [--] [<paths>]

Arguments:
  paths                         Optional path(s) to execute. Could be:
                                    - a dir (features/)
                                    - a feature (*.feature)
                                    - a scenario at specific line (*.feature:10)
                                    - all scenarios at or after a specific line (*.feature:10-*)
                                    - all scenarios at a line within a specific range (*.feature:10-20)
                                    - a scenarios list file (*.scenarios)

Options:
  -s, --suite=SUITE             Only execute a specific suite.
  -f, --format=FORMAT               How to format tests output. pretty is default.
                                Available formats are:
                                - progress  Prints one character per step.
                                - pretty  Prints the feature as is.
                                You can use multiple formats at the same time (multiple values allowed)
  -o, --out=OUT                 Write format output to a file/directory instead of
                                STDOUT (output_path). You can also provide different
                                outputs to multiple formats (multiple values allowed)
      --format-settings=FORMAT-SETTINGS  Set formatters parameters using json object.
                                Keys are parameter names, values are values. (multiple values allowed)
      --init                    Initialize all registered test suites.
```

```
    --lang=LANG                Print output in particular language.
    --name=NAME                Only executeCall the feature elements which match part of the given name or regex. (multiple values allowed)
    --tags=TAGS                Only executeCall the features or scenarios with tags matching tag filter expression. (multiple values allowed)
    --role=ROLE                Only executeCall the features with actor role matching a wildcard.
    --story-syntax             Print *.feature example.
                               Use --lang to see specific language.
-d, --definitions=DEFINITIONS  Print all available step definitions:
                               - use --definitions l to just list definition expressions.
                               - use --definitions i to show definitions with extended info.
                               - use --definitions 'needle' to find specific definitions.
                               Use --lang to see definitions in specific language.
    --append-snippets          Appends snippets for undefined steps into main context.
    --no-snippets              Do not print snippets for undefined steps after stats.
    --strict                   Passes only if all tests are explicitly passing.
    --rerun                    Re-run scenarios that failed during last execution.
    --stop-on-failure          Stop processing on first failed scenario.
    --dry-run                  Invokes formatters without executing the tests and hooks.
-p, --profile=PROFILE          Specify config profile to use.
-c, --config=CONFIG            Specify config file to use.
-v, --verbose                  Increase verbosity of exceptions.
-h, --help                     Display this help message.
    --config-reference         Display the configuration reference.
-V, --version                  Display this behat version.
-n, --no-interaction           Do not ask any interactive question.
    --colors                   Force ANSI color in the output. By default color support is guessed
                           based on your platform and the output if not specified.
    --no-colors                Force no ANSI color in the output.
```

Por lo tanto, en nuestro caso concreto, cuando se quiera llamar a la Feature *Signup*, habrá que ejecutar el siguiente comando:

```
:dummy$ bin/behat features/signup.feature
```

En un principio se comprobaban siempre todos los tests, pero a medida que se iba desarrollando se cayó en la cuenta de que, en cada test que se hacía al aplicar BDD, no era necesario probar cada funcionalidad desarrollada hasta el momento (la ejecución tarda bastante más). Por lo tanto, se decidió probar las funcionalidades individualmente.

El problema observado entonces era que, al probar una funcionalidad concreta, se seguían levantado tantos drivers como funcionalidades se tenían. Fué en este punto en el que procedió a optimizar la ejecución de las pruebas: en vez de levantar todos los drivers (los que se iban a usar y los que no), en cada ejecución se levantan solamente los drivers especificados en el fichero de properties. De este modo, se levantan exclusivamente los drivers que utiliza el Caso de Uso a probar.

11.3. Resultados

Por defecto, los resultados obtenidos a medida que se va desarrollando la aplicación son por terminal. Al ejecutar el comando especificado en el apartado anterior, Behat nos indica los escenarios de la funcionalidad que pasan con éxito (verde). Cuando un escenario falla, nos indica en rojo el escenario, el motivo y línea del fallo. De este modo, gracias a los resultados que se van obteniendo a medida que se prueba, el desarrollo con Behat es un desarrollo guiado.

Al tratarse de una respuesta por terminal como si se tratase de la respuesta de un comando del sistema, se tiene la posibilidad de redirigir la salida a un fichero `*.txt`.

Cuando se levanta el servidor de Selenium (y se especifica en el fichero de properties que se desea ejecutar la prueba con Firefox o Chrome), al ejecutar el comando por terminal, además de la información que éste nos habrá proporcionado (si no se redirige la salida), la aplicación hace las capturas de pantalla del estado del portal *dummy* al final de cada prueba y las guardará en la

carpeta correspondiente. De este modo se podrán consultar a posteriori los mensajes de error que el portal iba dando con cada prueba, entre otras cosas.

11.4. Objetivo de la prueba (web)

El objetivo de la prueba es el portal web *dummy* que se ha desarrollado. A lo largo del proceso se han tenido en cuenta los factores de:

1. Especificación.
2. Diseño.
3. Implementación.

11.4.1. Especificación

El modelo conceptual del portal *dummy* se especifica en el siguiente diagrama UML:

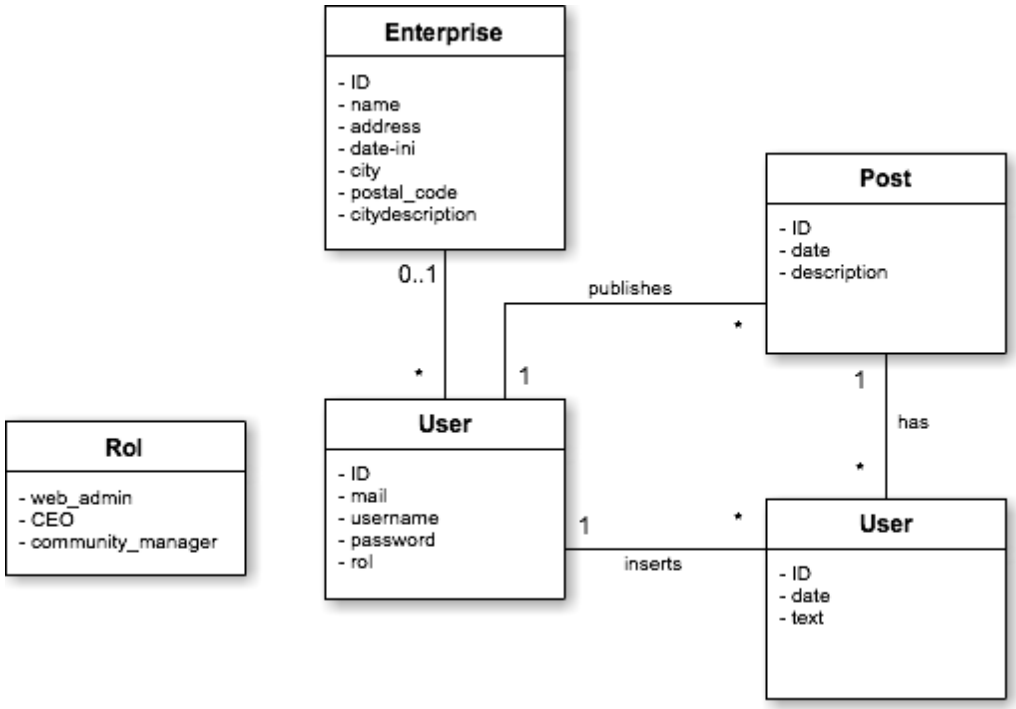


Imagen 21: Diagrama UML del portal

11.4.2. Diseño

En lo que al diseño del portal se refiere, se han tenido que tener en cuenta 3 temas:

1. Diseño de las pantallas.
2. Diseño de la navegación.
3. Diseño de la BBDD.

11.4.2.1. Pantallas

El resultado final se resume con las siguientes imágenes de las pantallas.

Como puede observarse, la página de login contiene los campos para introducir el nombre de usuario y la contraseña además del botón para hacer el login. También se tiene el botón de signup para el caso en el que el usuario no esté registrado.

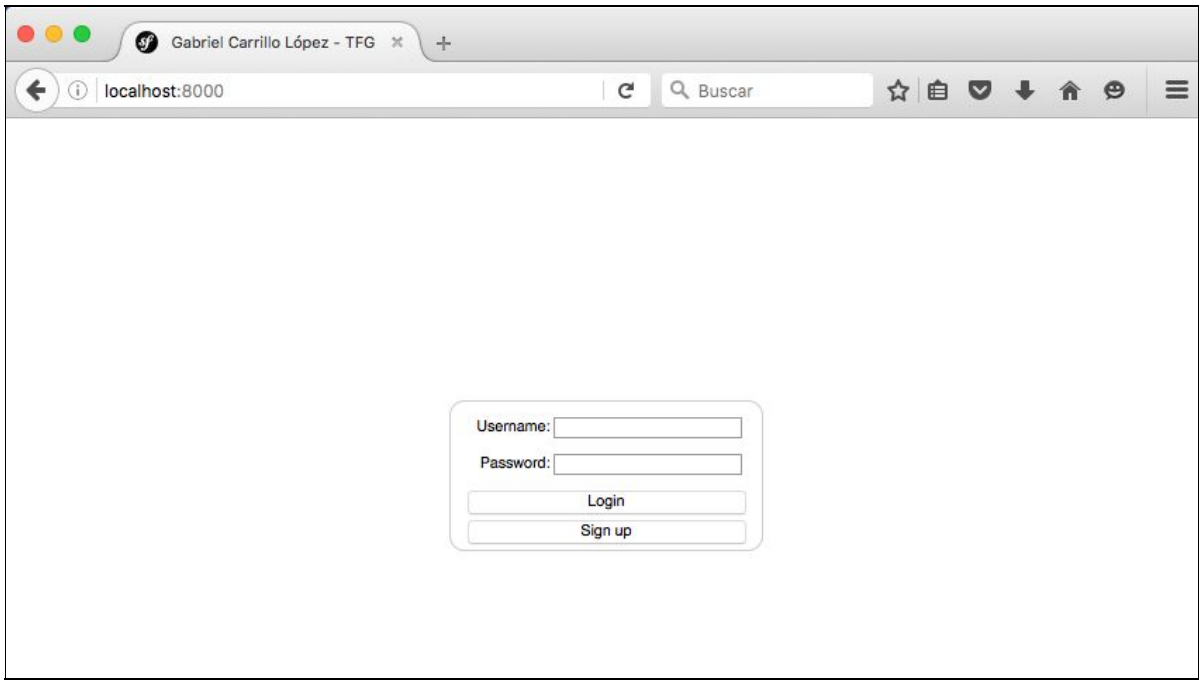


Imagen 22: Pantalla de login

Cuando un usuario introduce una combinación de nombre de usuario y contraseña incorrectos, la pantalla muestra el campo de mensajes de error con el error correspondiente..

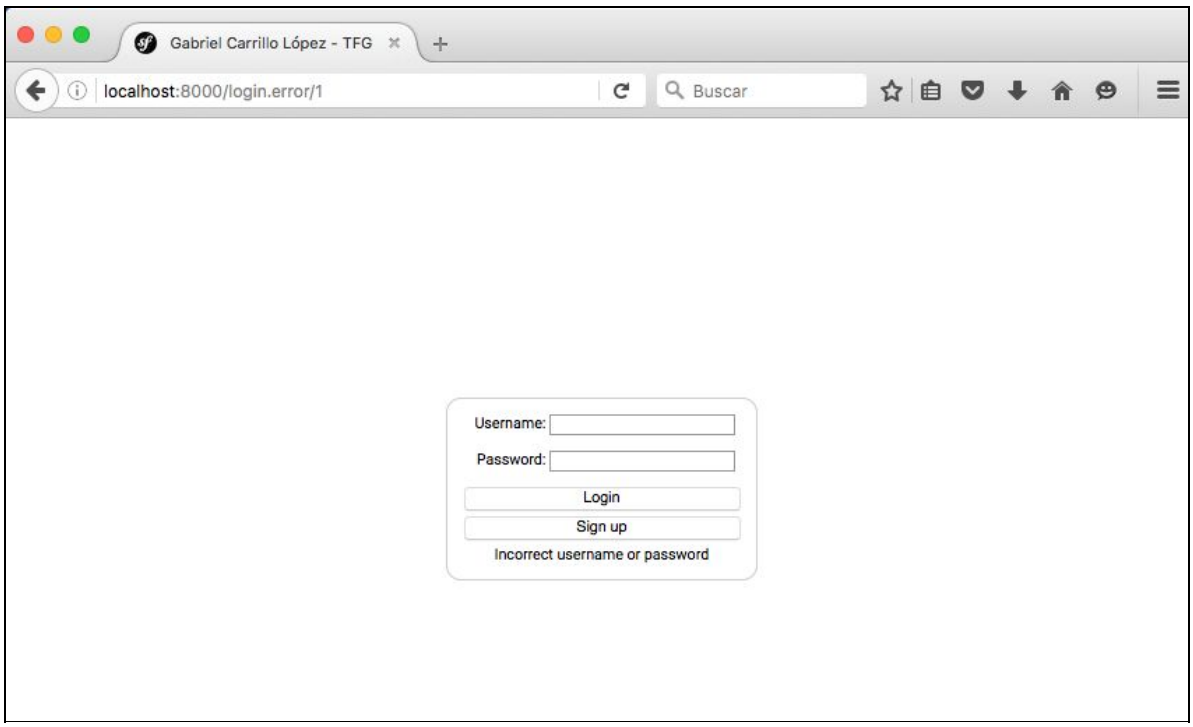


Imagen 23: Pantalla de login error

Si el usuario introduce el nombre de usuario y la contraseña correctos, accede al interior de la web. Aquí podrá hacer comentarios en la pestaña de Home, lugar en el que las empresas cuelgan sus anuncios.

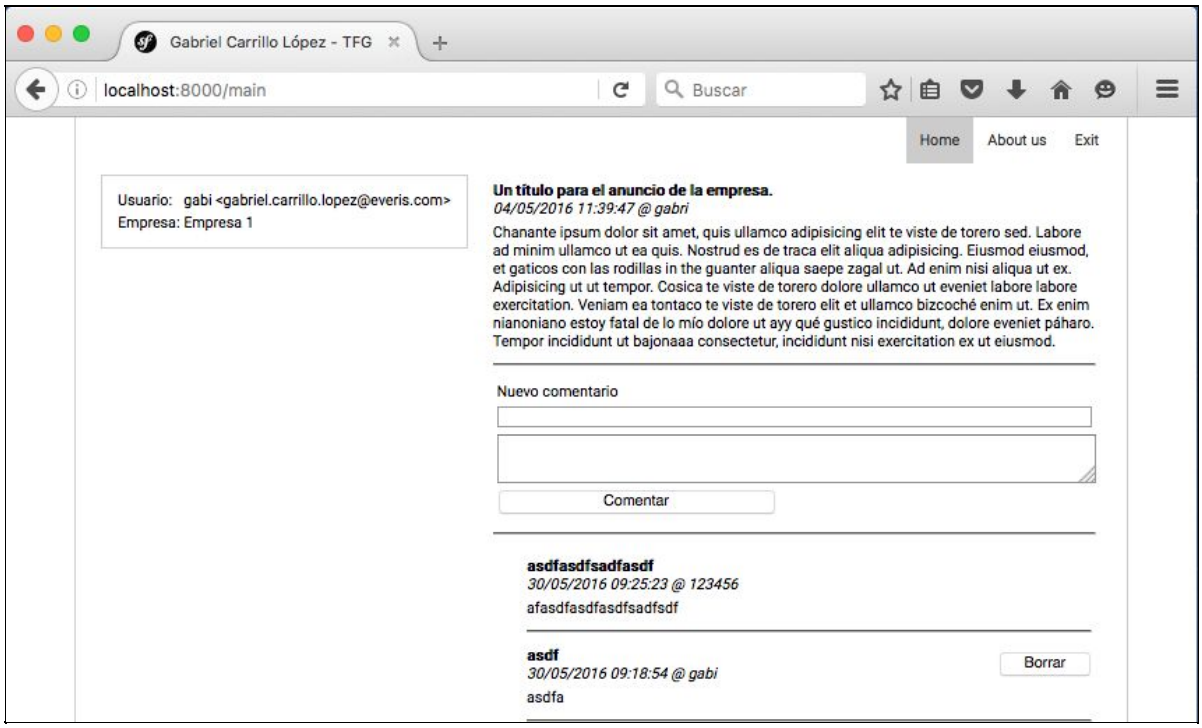


Imagen 24: Pantalla de Home

Si el usuario registrado intenta introducir un comentario sin título, la pantalla muestra el campo de mensajes de error con el error correspondiente.

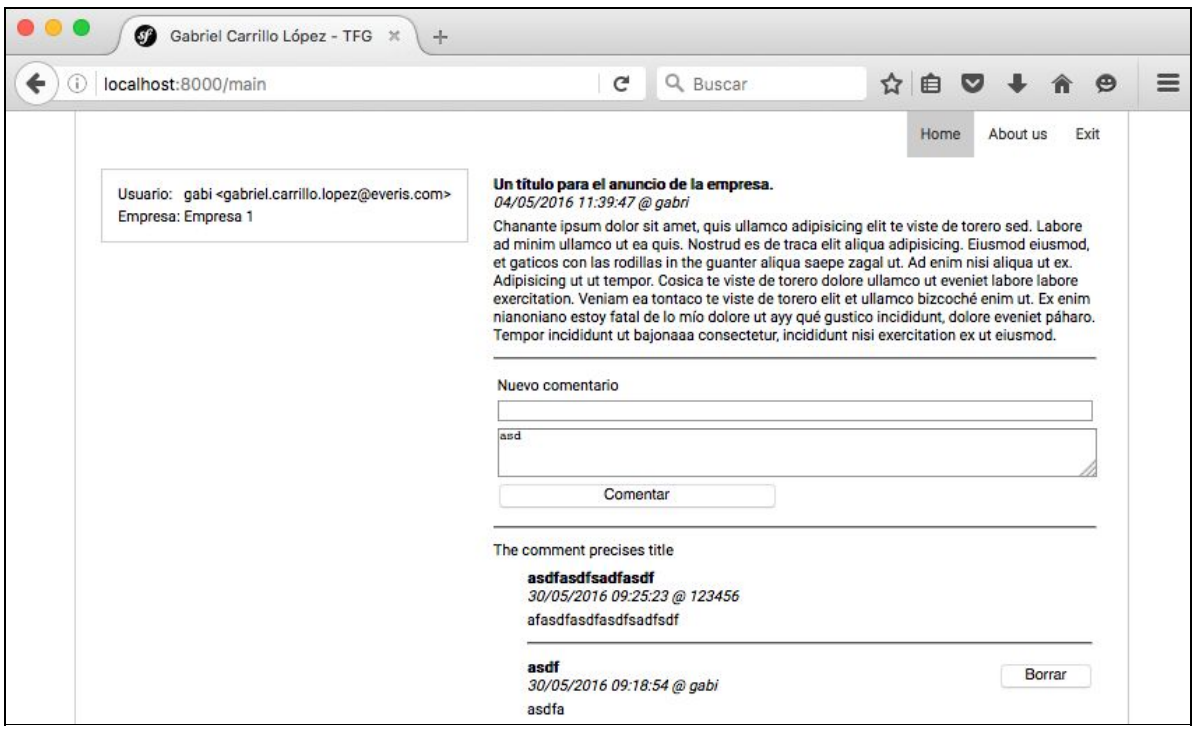


Imagen 25: Pantalla de Home error title

Si el usuario registrado intenta introducir un título de comentario sin cuerpo, la pantalla muestra el campo de mensajes de error con el error correspondiente.

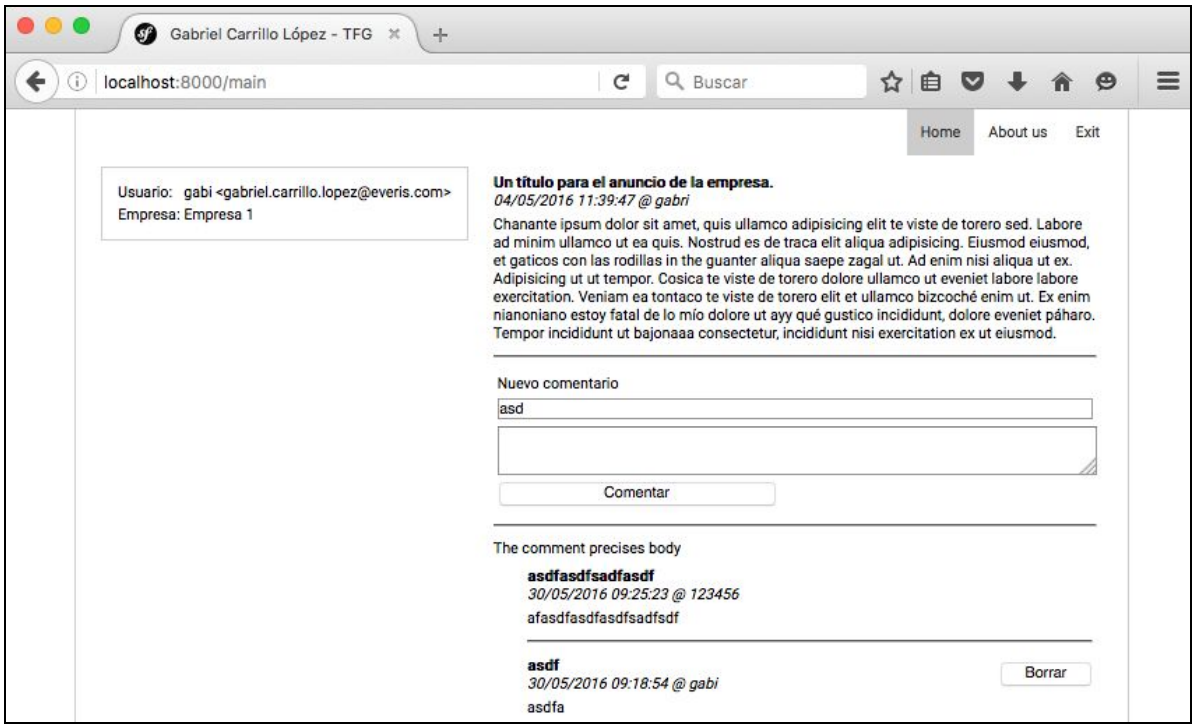


Imagen 26: Pantalla de Home error body

Si el usuario registrado intenta introducir un comentario con título y cuerpo, la pantalla muestra el comentario correspondiente y da al autor la posibilidad de borrarlo.

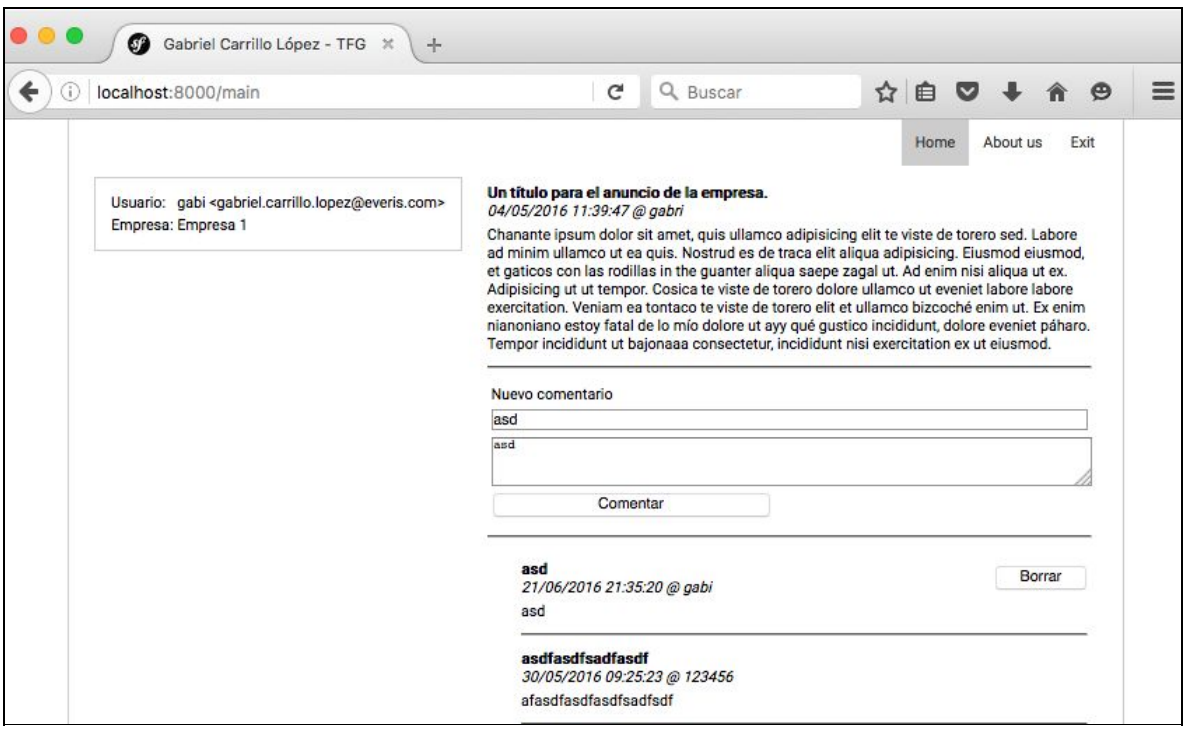


Imagen 27: Pantalla de Home insertar comentario

Si el usuario registrado intenta borrar un comentario suyo, la pantalla lanza un pop-up de confirmación.

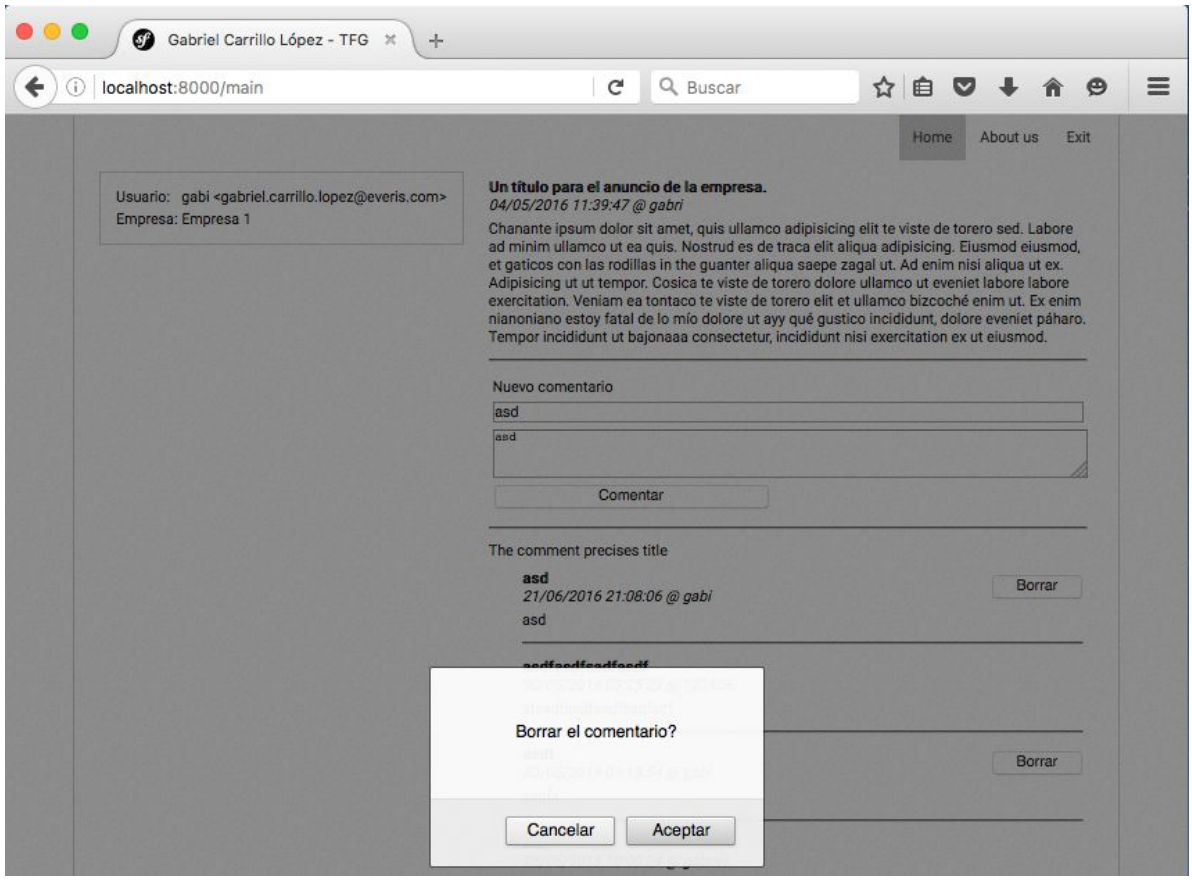


Imagen 28: Pantalla de Home borrar comentario

Si el usuario se registra correctamente, aparecerá en la página de bienvenida.

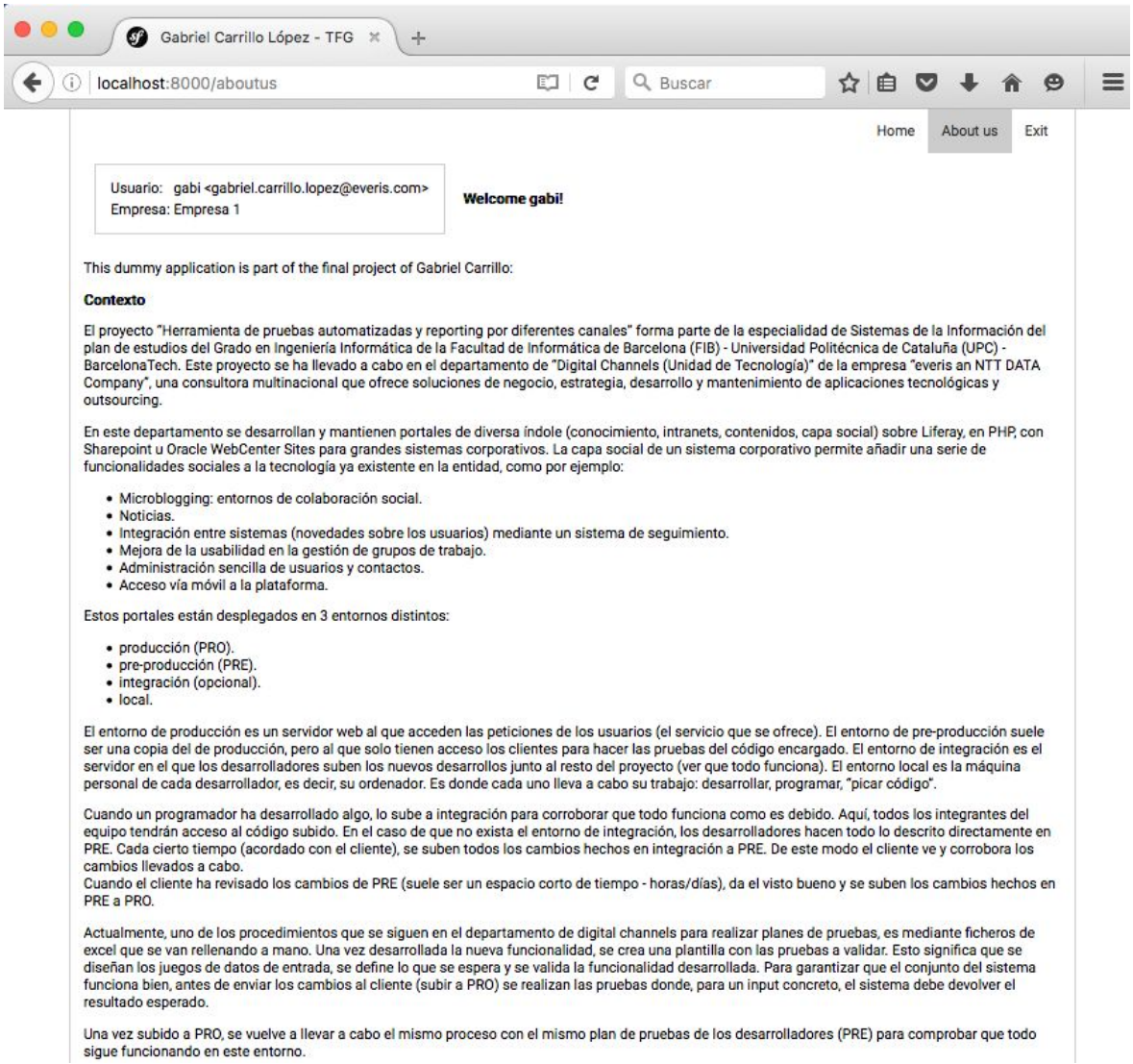


Imagen 29: Pantalla de About us

Si un usuario intenta registrars, aparecerá en la página de signup.

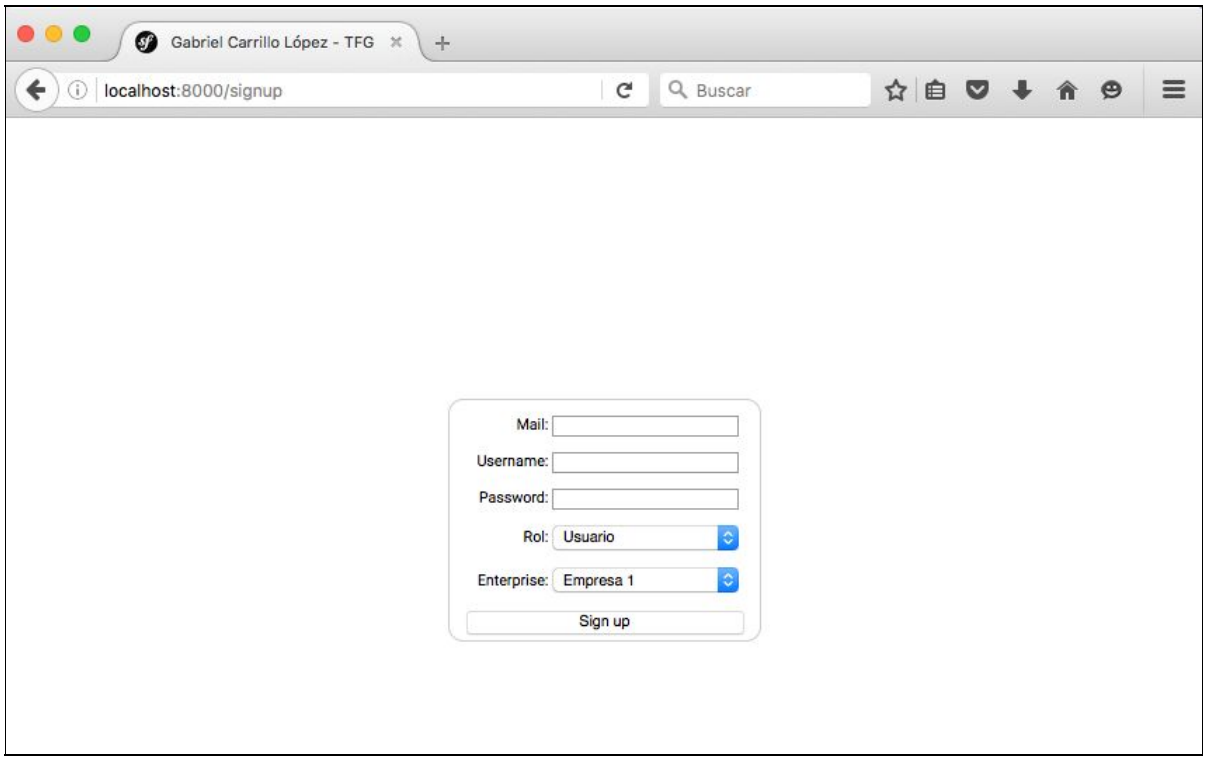


Imagen 30: Pantalla de signup

Si el usuario intenta registrarse con un e-mail inválido, la pantalla muestra el campo de mensajes de error con el error correspondiente.

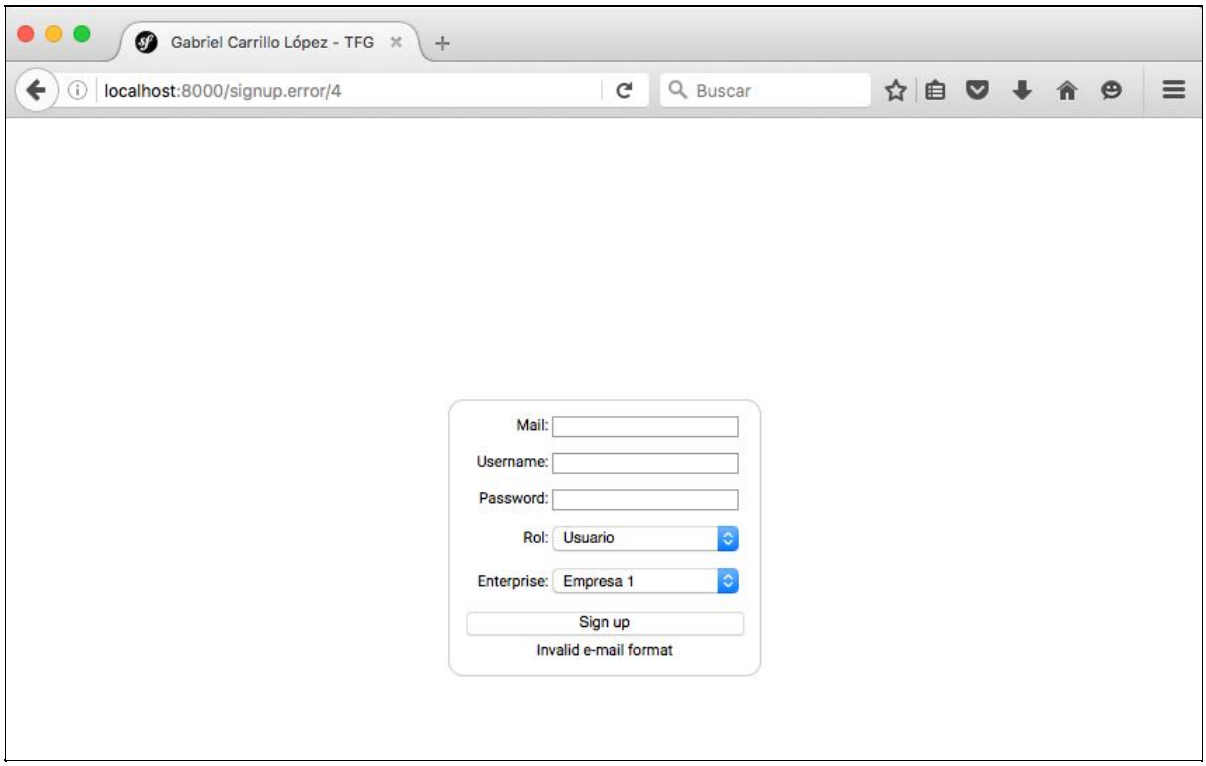


Imagen 31: Pantalla de signup error email

Si el usuario intenta registrarse con un nombre de usuario inválido, la pantalla muestra el campo de mensajes de error con el error correspondiente.

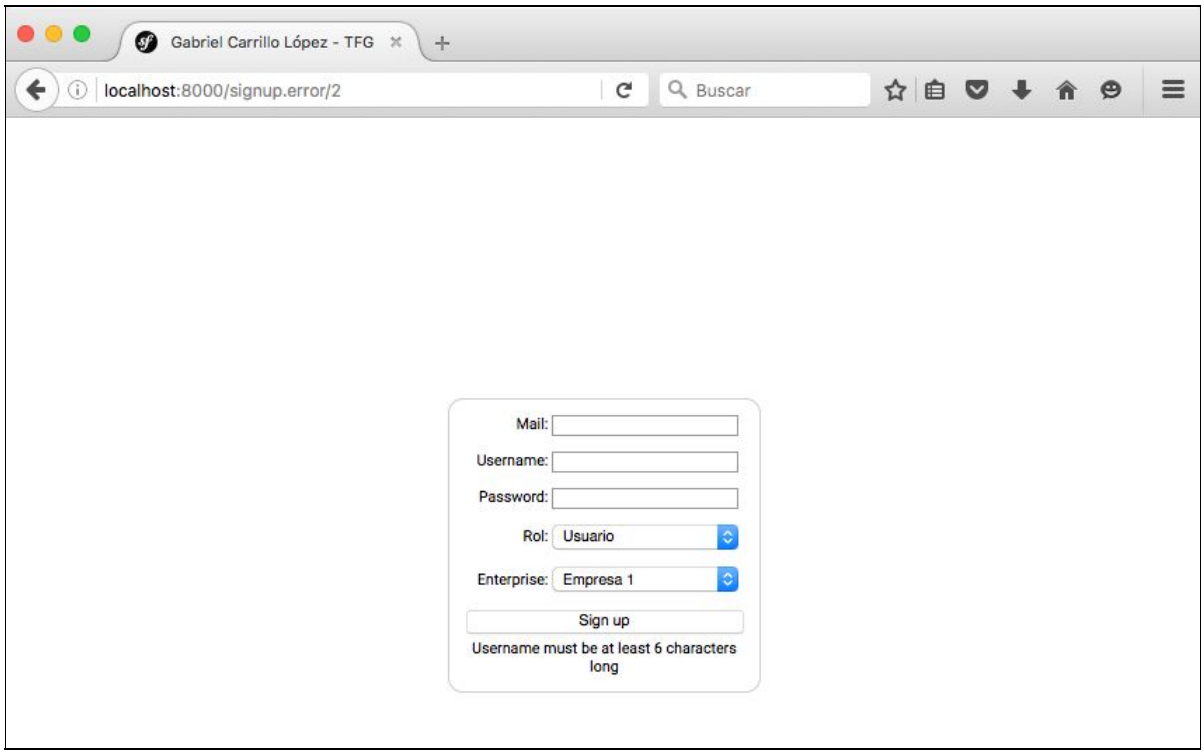


Imagen 32: Pantalla de signup error username

Si el usuario intenta registrarse con una contraseña inválida, la pantalla muestra el campo de mensajes de error con el error correspondiente.

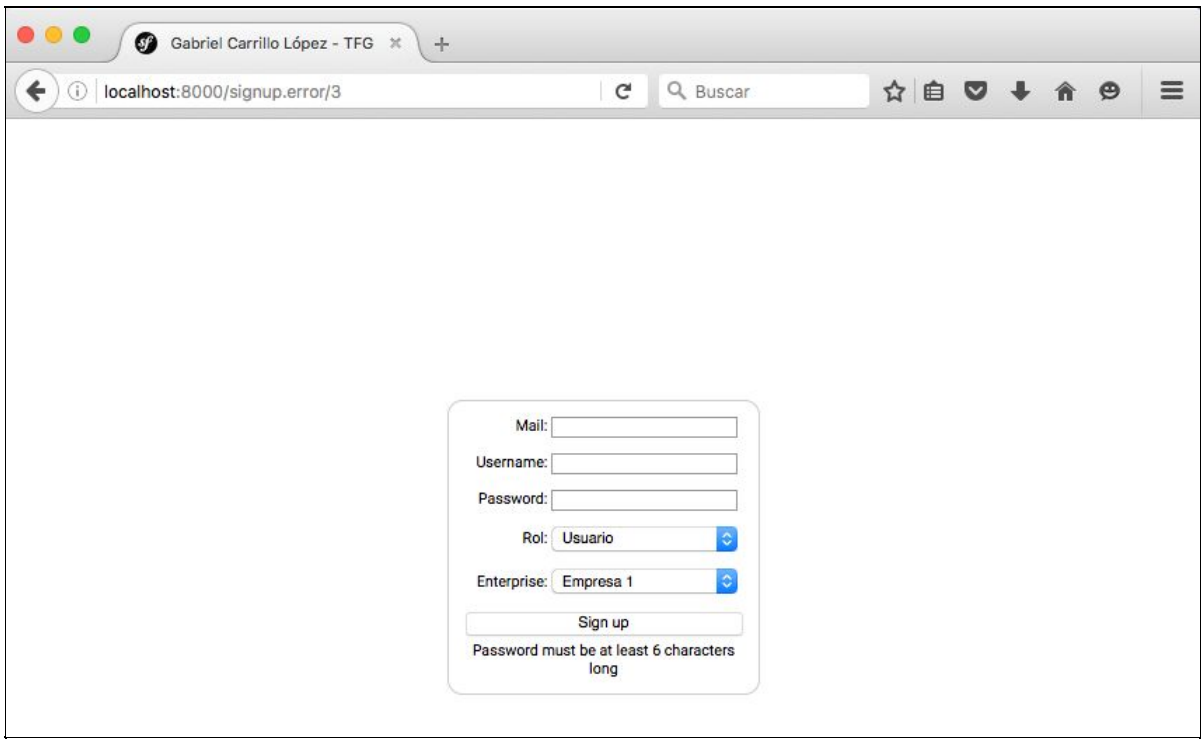


Imagen 33: Pantalla de signup error password

Si el usuario intenta registrarse sin alguno de los campos requeridos, la pantalla muestra el campo de mensajes de error con el error correspondiente.

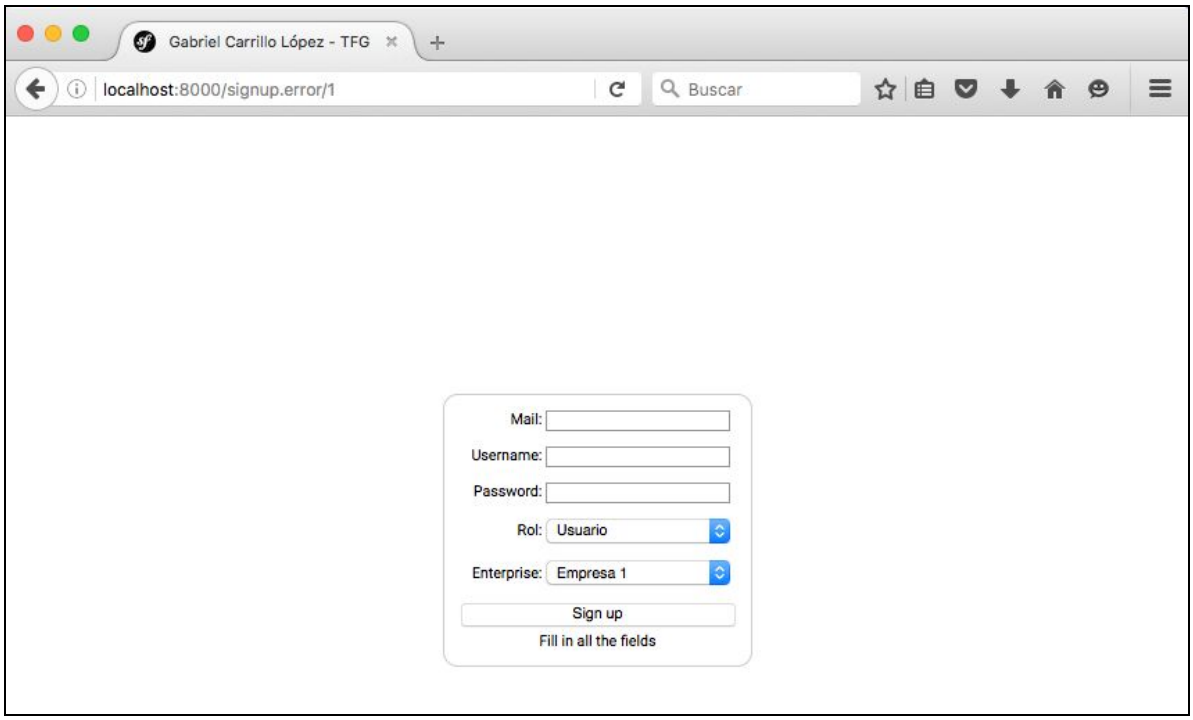


Imagen 34: Pantalla de signup error fields

11.4.2.2. Navegación

La navegación del portal web es como sigue:

11.4.2.2.1. Login

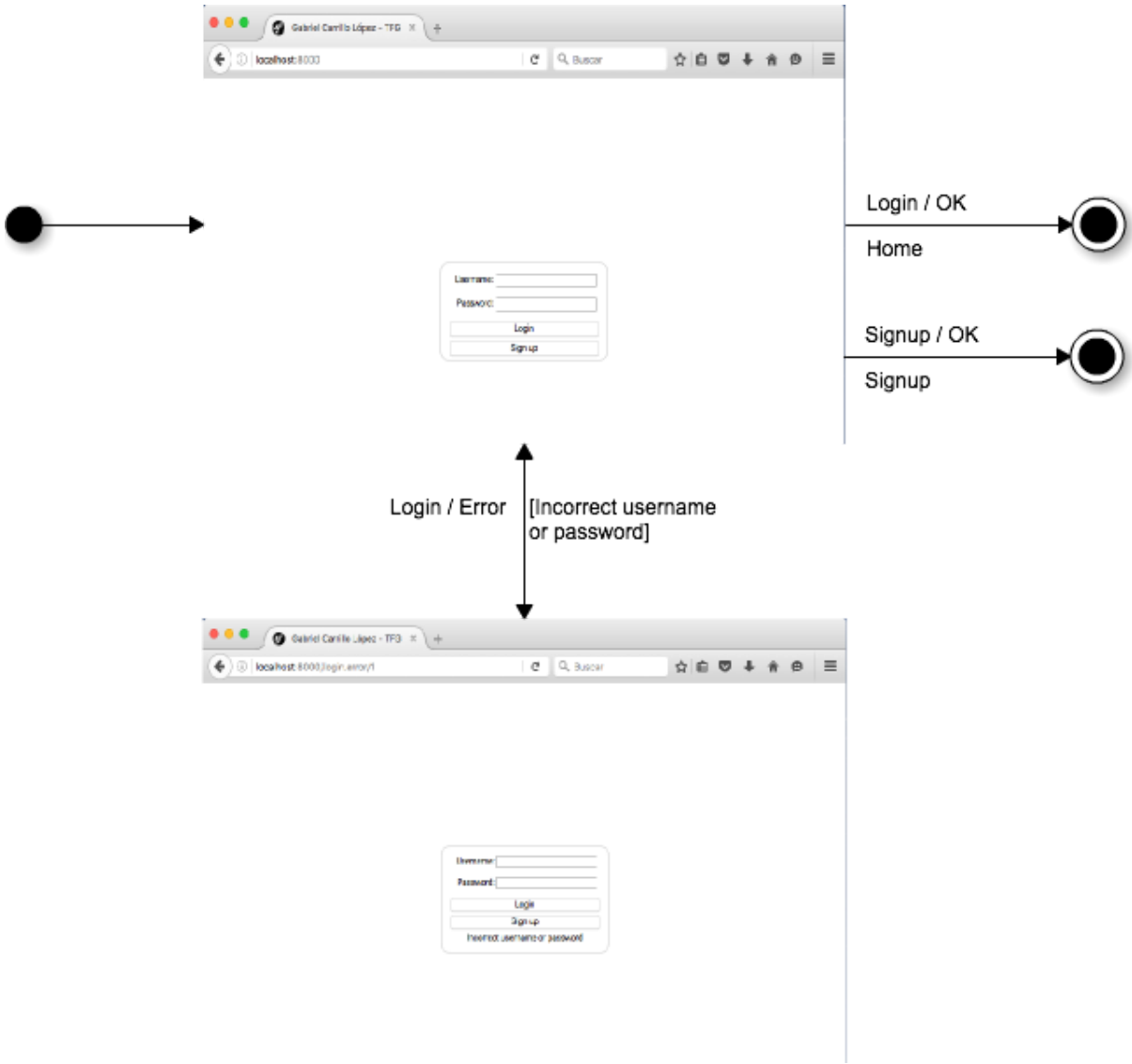


Imagen 35: Navegación página Login

11.4.2.2.2. Signup

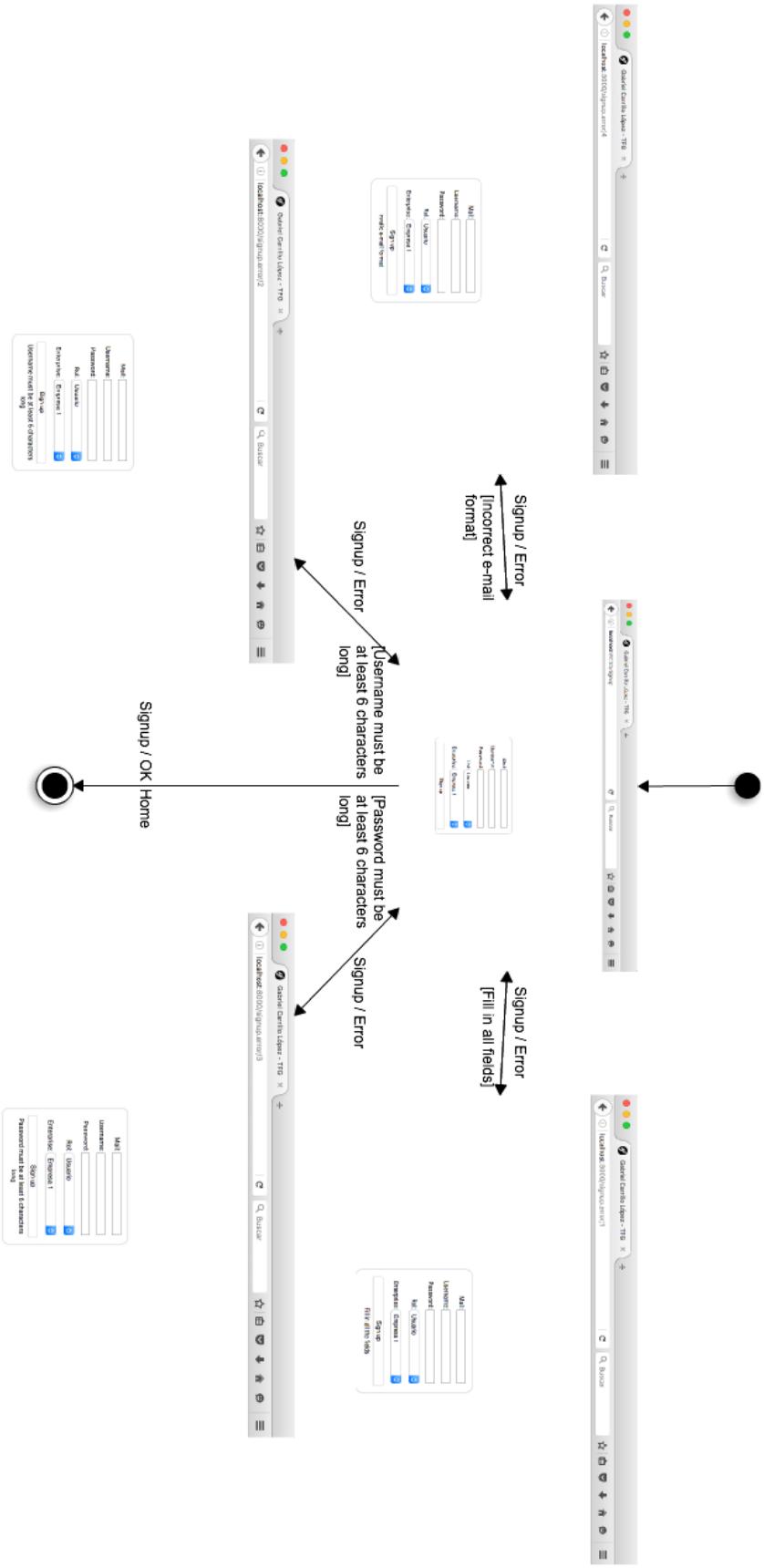


Imagen 36: Navegación página Signup

11.4.2.2.3. Home

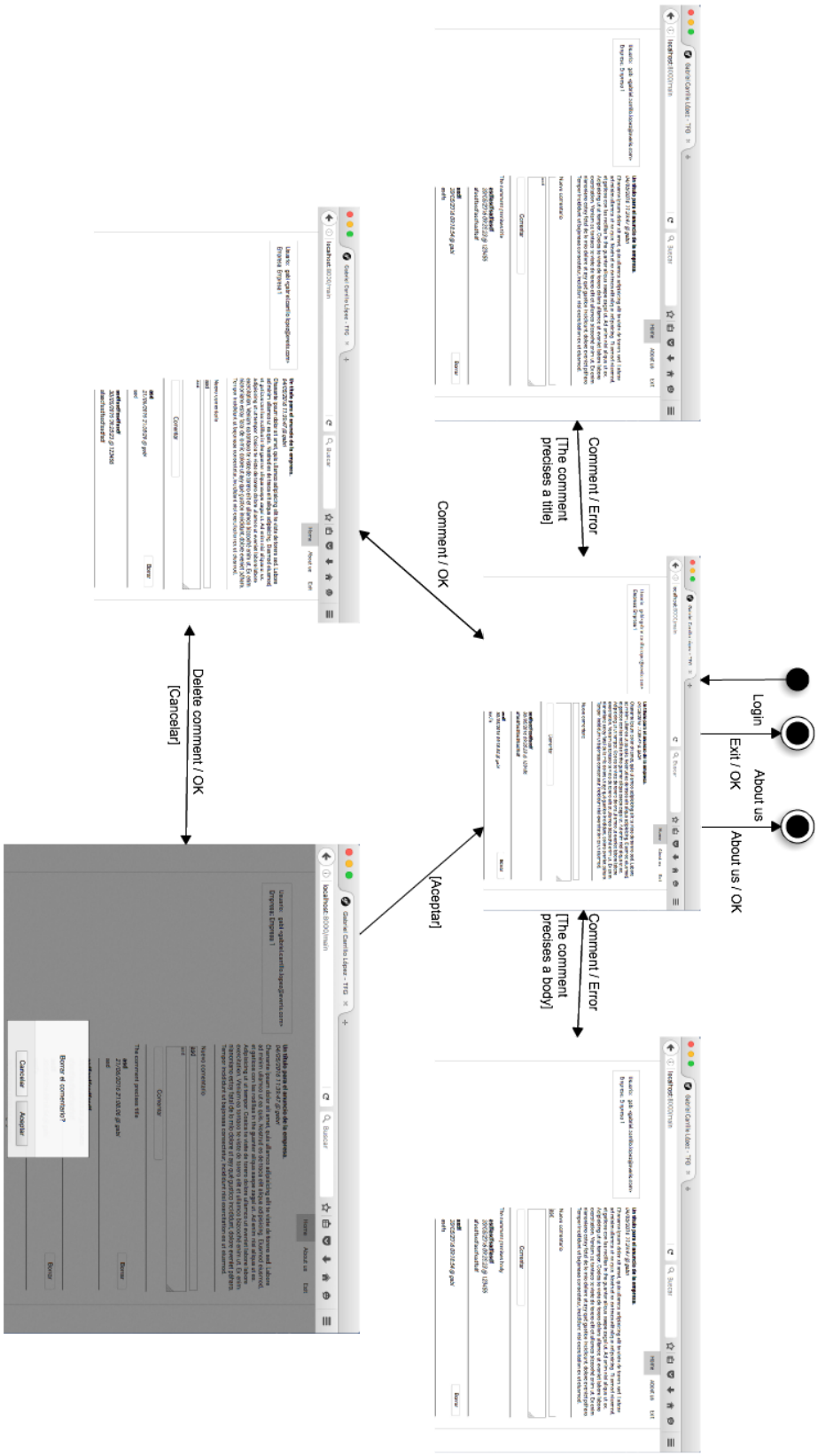


Imagen 37: Navegación página Home

11.4.2.2.4. About us

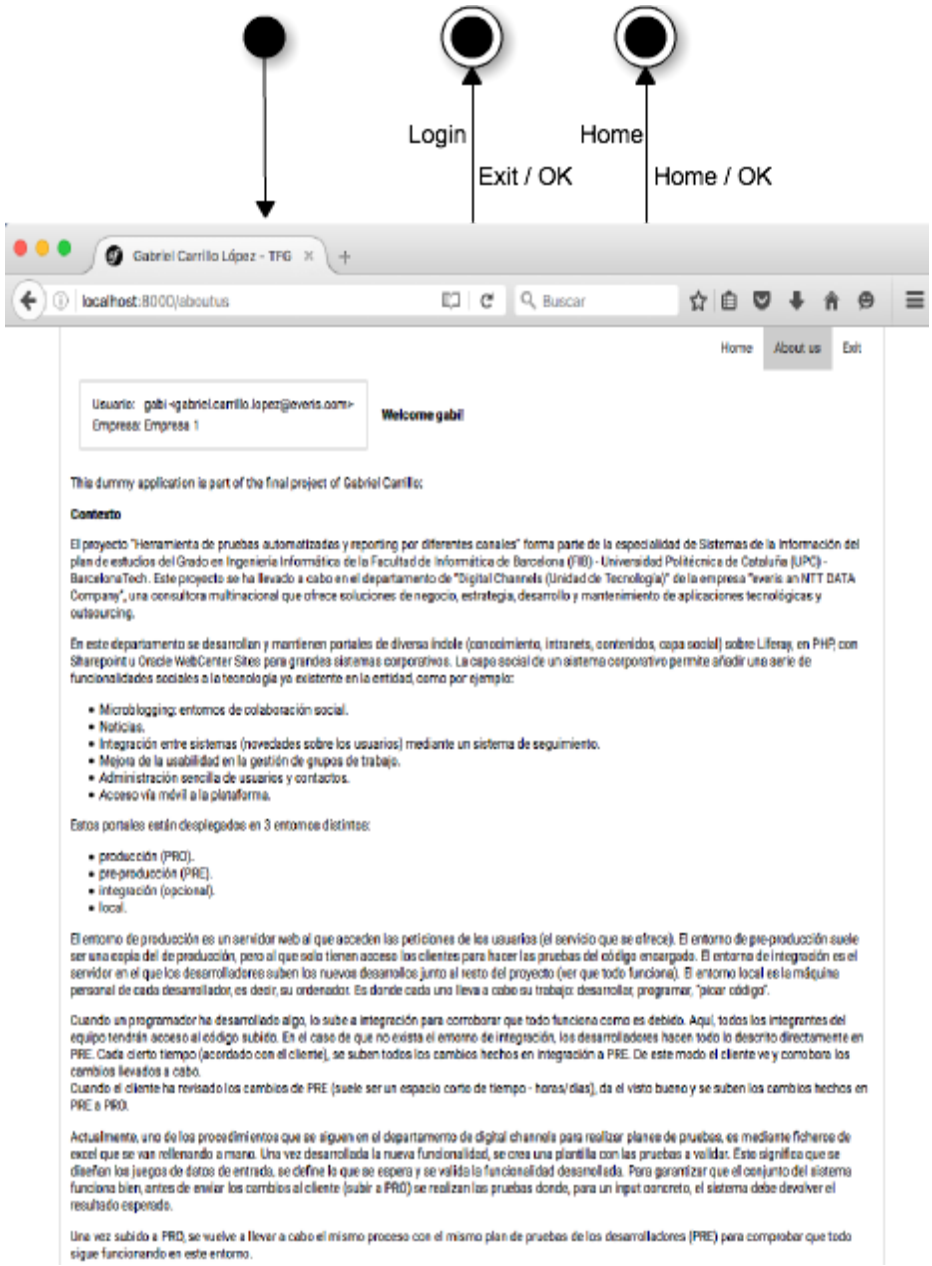


Imagen 38: Navegación página About us

11.4.2.3. BBDD

Para crear la BBDD según lo especificado en el esquema, se crearon los siguientes ficheros ORM:

1. Document.orm.yml.
2. Enterprise.orm.yml.
3. Post.orm.yml.
4. Rol.orm.yml.
5. User.orm.yml.

El contenido de, por ejemplo, el fichero *User*, sigue la siguiente estructura:

```yaml
AppBundle\Entity\User:
    type: entity
    repositoryClass: AppBundle\Entity\UserRepository
    table: user
    id:
        id:
            type: integer
            nullable: false
            options:
                unsigned: true
            generator:
                strategy: IDENTITY
    fields:
        mail:
            type: string
            length: 255
            nullable: false
        username:
            type: string
            length: 50
            nullable: false
        password:
            type: string
            length: 16
            nullable: false
    manyToOne:
        fk_enterpriseId:
            targetEntity: AppBundle\Entity\Enterprise
            joinColumn:
                name: enterpriseId
                referencedColumnName: id
                nullable: false
        fk_rolId:
            targetEntity: AppBundle\Entity\Rol
            joinColumn:
                name: rolId
                referencedColumnName: id
                nullable: false
    lifecycleCallbacks: {  }
```

11.4.3. Implementación

En este apartado se especifican los servicios utilizados en el desarrollo.

11.4.3.1. Servicios

Al desarrollar el portal, se decidió definir las siguientes clases como servicios:

1. User.
2. Post.
3. Session.
4. Enterprise.
5. Rol.

Esto es así para permitir el acceso a las mismas desde cualquier punto de la aplicación, inyectar dependencias a los servicios en su definición (nosotros inyectamos el contenedor de la aplicación para que el servicio sea capaz de comunicarse con la aplicación y la aplicación con todos los servicios de forma transversal) y flexibilizar su instanciación (para poder extrapolar la aplicación en futuros evolutivos, para por ejemplo añadir un servicio de correo). La forma de especificar estos servicios es a través del fichero services.yml con la siguiente estructura:

```
services:
   model.user:
       class: AppBundle\Model\UserModel
       arguments: [@doctrine.orm.entity_manager, @service_container]

   model.post:
       class: AppBundle\Model\PostModel
       arguments: [@doctrine.orm.entity_manager, @service_container]

   model.session:
       class: AppBundle\Model\SessionModel
       arguments: [@doctrine.orm.entity_manager, @service_container]

   model.enterprise:
       class: AppBundle\Model\EnterpriseModel
       arguments: [@doctrine.orm.entity_manager, @service_container]

   model.rol:
       class: AppBundle\Model\RolModel
       arguments: [@doctrine.orm.entity_manager, @service_container]
```

12. Ejecución pruebas (explicación)

Para poder llevar a cabo una demostración del proyecto hará falta levantar el portal dummy. Para ello, antes hay que levantar el servidor de la BBDD a la que el portal accede para la persistencia de los datos. En nuestro caso específico lo hacemos con XAMPP, dándole al botón de "Start" al servidor "MySQL Database".

A continuación, para levantar el portal hay que abrir el terminal y ejecutar el siguiente comando:

```
:dummy$ app/console server:run
```

De este modo ya se puede acceder al portal *dummy* desde cualquier navegador introduciendo la siguiente URL:

```
http://localhost:8000
```

Anteriormente se han mencionado los distintos Drivers y Casos de Uso utilizados en el proyecto, por lo que en el caso de querer probar el Caso de Uso *Sign up* con Goutte, habría que introducir en el fichero de properties el número "**1**" en el espacio reservado para escoger el *Driver*, y la letra "**b**" en el espacio reservado para escoger el *Caso de Uso*. A continuación, habría que ir al terminal y ejecutar el siguiente comando:

```
:dummy$ bin/behat features/signup.feature
```

De este modo se observarían los resultados de la prueba por terminal y no se harían capturas de pantalla.

Si se quisiese redirigir la salida, habría que añadir al final del comando el destino como sigue:

```
:dummy$ bin/behat features/signup.feature > /Applications/XAMPP/htdocs/dummy/web/
        tests_output/signupTestAction/salidaSignup.txt
```

De este modo siempre podríamos acceder a los resultados de la prueba en la ruta especificada, pero sin capturas de pantalla. En el caso probado, el documento contiene la información como sigue:

```
Feature: Sign up
  In order to sign up
  As a new user
  I need to be able to provide the necessary information to the system

  Rules:
  - Mail contains an @
  - Username with > 6 characters
  - Password with > 6 characters
  - Mail not containing an @ returns error "Invalid e-mail format"
  - Username with <= 5 characters returns error "Username must be at least 6 characters long"
  - Password with <= 5 characters returns error "Password must be at least 6 characters long"

  Scenario: Sign up to the system with an invalid e-mail                    # features/signup.feature:14
    Given I am not registered                                               # FeatureContext::iAmNotRegistered()
    When I try to "sign" to the system with an e-mail not containing an @ # FeatureContext::iTryToToTheSystemWithAnEMailNotContainingAn()
    Then I should see the signup error message "Invalid e-mail format" # FeatureContext::iShouldSeeTheSignupErrorMessage()

  Scenario: Sign up to the system with an invalid username                    # features/signup.feature:19
    Given I am not registered                                                 # FeatureContext::iAmNotRegistered()
    When I try to "sign" to the system with a username shorter than 6 characters
                                                                              # FeatureContext::iTryToToTheSystemWithAUsernameShorterThanCharacters()
    Then I should see the signup error message "Username must be at least 6 characters long"
                                                                              # FeatureContext::iShouldSeeTheSignupErrorMessage()

  Scenario: Sign up to the system with an invalid password                    # features/signup.feature:24
    Given I am not registered                                                 # FeatureContext::iAmNotRegistered()
    When I try to "sign" to the system with a password shorter than 6 characters
                                                                              # FeatureContext::iTryToToTheSystemWithAPasswordShorterThanCharacters()
    Then I should see the signup error message "Password must be at least 6 characters long"
                                                                              # FeatureContext::iShouldSeeTheSignupErrorMessage()

  Scenario: Sign up to the system without some of the required data          # features/signup.feature:29
    Given I am not registered                                                 # FeatureContext::iAmNotRegistered()
    When I try to "sign" to the system without an e-mail, username or password    #
FeatureContext::iTryToToTheSystemWithoutAnEMailUsernameOrPassword()
    Then I should see the signup error message "Fill in all the fields"       # FeatureContext::iShouldSeeTheSignupErrorMessage()
```

```
  Scenario: Sign up to the system with all the required data                    # features/signup.feature:34
    Given I am not registered                                                   # FeatureContext::iAmNotRegistered()
    When I try to "sign" to the system with the e-mail "gabriel.carrillo.lopez@everis.com", the username "Wenceslao", the password "999999999",
the rol "Administrador" and the enterprise "Empresa 1"
                                  # FeatureContext::iTryToToTheSystemWithTheEMailTheUsernameThePasswordTheRolAndTheEnterprise()
    Then I should see the welcome title "Welcome Wenceslao!"                    # FeatureContext::iShouldSeeTheWelcomeTitle()

5 escenarios (5 pasaron)
15 pasos (15 pasaron)
0m2.98s (15.54Mb)
```

Como puede comprobarse, se ha testeado la *Feature* (funcionalidad, Caso de Uso) *Sign up*, que consta de 5 escenarios:

- Sign up to the system with an invalid e-mail.
- Sign up to the system with an invalid username.
- Sign up to the system with an invalid password.
- Sign up to the system without some of the required data.
- Sign up to the system with all the requested data.

Al final del documento se indica que han pasado los 5 escenarios con éxito en 2.98 segundos.

En el caso de querer probar el Caso de Uso *Comment*, habría que usar el *Driver* Selenium2 con Chrome, por lo que habría que introducir en el fichero de properties el número "**4**" en el espacio reservado para escoger el *Driver*, y la letra "**e**" en el espacio reservado para escoger el *Caso de Uso*. El siguiente paso sería levantar el servidor Selenium2, para lo que habría que ir al terminal y ejecutar el siguiente comando:

```
:dummy$ java -jar selenium-server-standalone-2.53.0.jar
```

Siendo el último parámetro el nombre del fichero `*.jar` descargado con anterioridad.
A continuación, habría que ir al terminal y ejecutar el siguiente comando:

```
:dummy$ bin/behat features/comment.feature
```

De este modo se observarían los resultados de la prueba por terminal y se visualizarían las pruebas que se irían haciendo en un buscador. Además se harían capturas de pantalla de lo visualizado y se guardarían automáticamente en la carpeta siguiente:

```
/Applications/XAMPP/htdocs/dummy/web/tests_output/commentTestAction/screenshots/
```

Si se quisiese redirigir la salida, habría que añadir al final del comando el destino como sigue:

```
:dummy$ bin/behat features/comment.feature > /Applications/XAMPP/htdocs/dummy/web/
        tests_output/commentTestAction/salidaComment.txt
```

De este modo siempre podríamos acceder a los resultados de la prueba en la ruta especificada además de a las capturas de pantalla.

En el caso probado, el documento contiene la información como sigue:

```
Feature: Comment
  In order to comment a post
  As a logged user
  I need to be able to introduce a post title and body and click the "Comentar" button

  Rules:
  - I'm a logged user

  Scenario: Comment with a title                                                # features/comment.feature:9
    Given I am logged as username "gabriel" with password "gabriel"             # FeatureContext::iAmLoggedAsUsernameWithPassword()
    When I try to surf to the "Home" tab                                        # FeatureContext::iTryToSurfToTheTab()
    And I try to comment the post with the title "Comment with a title"         # FeatureContext::iTryToCommentThePostWithTheTitle()
    Then I should see the comment error message "The comment precises body"     # FeatureContext::iShouldSeeTheCommentErrorMessage()

  Scenario: Comment with a body                                                 # features/comment.feature:15
    Given I am logged as username "jose" with password "jose"                   # FeatureContext::iAmLoggedAsUsernameWithPassword()
    When I try to surf to the "Home" tab                                        # FeatureContext::iTryToSurfToTheTab()
    And I try to comment the post with the body "Comment with a body"           # FeatureContext::iTryToCommentThePostWithTheBody()
    Then I should see the comment error message "The comment precises title"    # FeatureContext::iShouldSeeTheCommentErrorMessage()
```

```
Scenario: Comment with a title and a body                                  # features/comment.feature:21
  Given I am logged as username "123456" with password "123456"            # FeatureContext::iAmLoggedAsUsernameWithPassword()
  When I try to surf to the "Home" tab                                     # FeatureContext::iTryToSurfToTheTab()
  And I try to comment the post with the title "Comment with a title and a body" and the body "Body of the comment with a title and a body"
                                                                           # FeatureContext::iTryToCommentThePostWithTheTitleAndTheBody()
  Then I should see the comment with the title "Comment with a title and a body" # FeatureContext::iShouldSeeTheCommentWithTheTitle()

Scenario: Delete a post                                                    # features/comment.feature:27
  Given I am logged as username "123456" with password "123456"            # FeatureContext::iAmLoggedAsUsernameWithPassword()
  When I try to surf to the "Home" tab                                     # FeatureContext::iTryToSurfToTheTab()
  And I commented with the title "Comment with a title and a body" and the body "Body of the comment with a title and a body"
                                                                           # FeatureContext::iCommentedWithTheTitleAndTheBody()
  And I try to delete the post with title "Comment with a title and a body" # FeatureContext::iTryToDeleteThePostWithTitle()
  Then I should not see the post with title "Comment with a title and a body" # FeatureContext::iShouldNotSeeThePostWithTitle()

4 escenarios (4 pasaron)
17 pasos (17 pasaron)
0m16.13s (12.99Mb)
```

Como puede comprobarse, se ha testeado la Feature *Comment*, que consta de 4 escenarios, con sus respectivas capturas de pantalla:

- **Comment with a title**

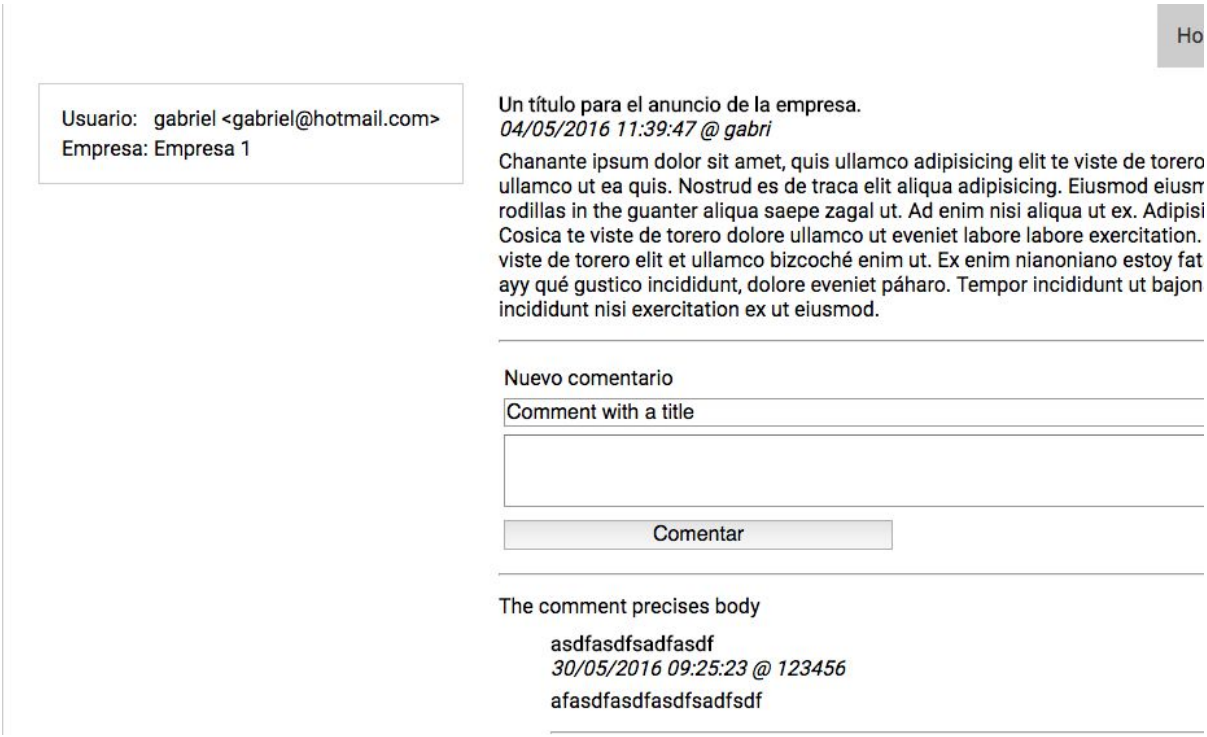


Imagen 39: iShouldSeeTheCommentErrorMessage--The comment precises body--2016/06/20-13.58.34

- **Comment with a body**

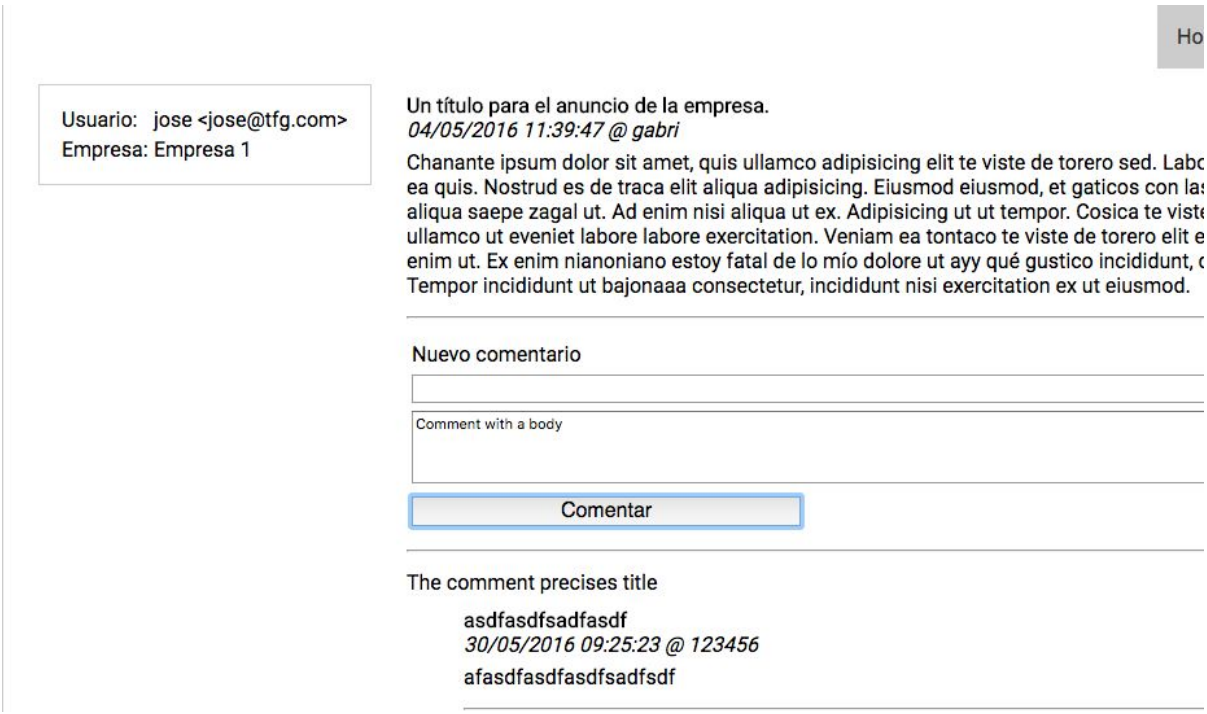


Imagen 40: iShouldSeeTheCommentErrorMessage--The comment precises title--2016/06/20-13.58.38

- **Comment with a title and a body**

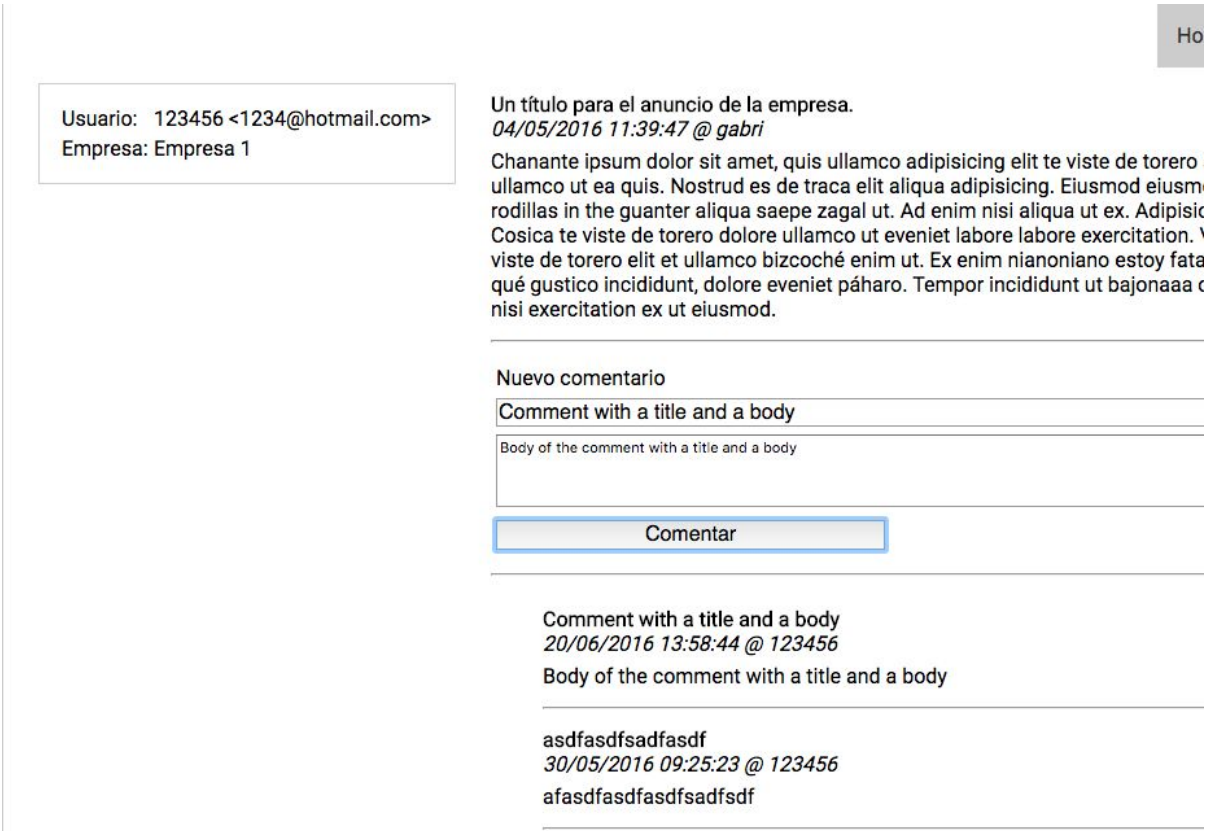


Imagen 41: iShouldSeeTheCommentWithTheTitle--Comment with a title and a body--2016/06/20-13.58.41

- **Delete post**

Imagen 42: iShouldNotSeeThePostWithTitle--Comment with a title and a body--2016/06/20-13.58.46

Al final del documento se indica que han pasado los 4 escenarios con éxito en 16.135 segundos.

Este es un buen ejemplo para comparar los tiempos entre usar un Emulador de navegadores sin interfaz gráfica (2.98 segundos) o un Controlador de navegadores (16.135 segundos). Pese a tratarse de Casos de Uso distintos, la proporción de los tiempos es bastante parecida, puesto que la Feature Signup con el Controlador de navegadores tarda 17.06 segundos.

13. Sostenibilidad

Las tres dimensiones de la sostenibilidad son los puntos de vista económico, social y ambiental del proyecto en su entorno. Se ha analizado la matriz de sostenibilidad resultando en la siguiente puntuación:

Sostenibilidad	Ambiental	Económico	Social	Rango Sost.
PPP	**8**:10	**7**:10	**8**:10	**23**:30

Tabla 9: Detalle de la matriz de sostenibilidad

13.1. Estudio de impacto Ambiental

Para minimizar el impacto ambiental, en vez de comprar y usar hardware y software nuevo, se ha decidido reutilizar los recursos a los que el estudiante ya tenía acceso en su uso diario cotidiano. Además, cuando se han precisado nuevas herramientas, se ha optado por usar las que tenían soporte online para evitar el uso de memorias portátiles. Esto reduce el número de horas invertidas en mantener actualizadas las copias del proyecto en los distintos entornos de trabajo y, por lo tanto, aumenta el número de horas invertidas en avanzar el proyecto. Esto quiere decir que se potencia el trabajo eficiente.

Los terminales con los que se trabaja se ponen en reposo (pantalla primero y CPU después) al minuto de inactividad, por lo que si se está llevando en paralelo alguna actividad que requiera de la atención del estudiante durante más de 60 segundos, se ahorra energía.

Anteriormente a la realización de este proyecto, el trabajo se llevaba a cabo de forma manual: se rellenaban un fichero de excel con las pruebas a realizar, los resultados esperados y los resultados obtenidos para cada uno de los entornos. Esto implicaba una inversión de tiempo mucho más elevada para realizar la misma cantidad de trabajo, por lo que se consumían los mismos recursos durante un periodo más prolongado de tiempo. Esto implicaba un aumento del consumo de energía.

13.2. Estudio de impacto Económico

El coste de la realización del proyecto se estima en 24.856,6 € (IVA incluido).

La mejora de la solución propuesta es temporal, puesto que al automatizar las tareas de testing, se realizan en un tiempo mucho menor. Esto implica que los proyectos se pueden finalizar en menos tiempo, por lo que se centra el esfuerzo de los trabajadores en otras tareas. Al abordar más proyectos en menos tiempo, se ahorra dinero a la empresa. La cantidad ahorrada es proporcional al número de horas invertidas en las pruebas respecto a las totales del proyecto.

13.3. Estudio del impacto Social

El desarrollo del proyecto ha aportado al alumno la visión empresarial de los procesos actuales de desarrollo de software. Esto ha proporcionado la capacidad de identificar las posibilidades y necesidades de automatización.

Previo al desarrollo del proyecto, al llevarse el testeo a mano, se ejecutaba una tarea repetitiva que al ser llevada a cabo en un sedentarismo postural, podía causar fatiga laboral y mental en el trabajador[22]. La mejora social del equipo y la eficiencia económica y ambiental del proyecto lo convierte en un proyecto de relevancia para las partes.

[22] [UCM16]

14. Conclusiones

Pese a haberse cumplido con los objetivos tecnológicos y empresariales, ha faltado la integración final con TestLink. Cabe mencionar que desde el principio se había definido como funcionalidad opcional pero que finalmente fue eliminada por la modificación sufrida en la planificación temporal inicial. De no haber sido así, se hubiese completado el ciclo total de la obtención de resultados en el proceso de automatización, pero gracias a la modificación se ha finalizado el proyecto en el periodo establecido.

También hay que destacar la importancia del correcto diseño de los Casos de Uso y de la especificación sobre cómo se desea obtener los resultados. Esto es así por la gran diferencia en los tiempos de ejecución al usar un Emulador de navegadores sin interfaz gráfica (muy rápida) o un Controlador de navegadores (casi 10x más lento). Nuestro portal *dummy* es muy sencillo y las pruebas no requieren demasiados escenarios, pero en portales más complejos puede ser crucial.

Previo al desarrollo del proyecto, al llevarse el testeo a mano, se ejecutaba una tarea repetitiva que al ser llevada a cabo en un sedentarismo postural, podía causar fatiga laboral y mental en el trabajador. La mejora social del equipo y la eficiencia económica y ambiental del proyecto lo convierte en un proyecto de relevancia para las partes.

La solución propuesta, pese a ser mejorable, ha estudiado y propuesto soluciones a los aspectos más relevantes para la empresa en la que se ha desarrollado. La solución utiliza los conocimientos adquiridos, en su mayoría, a lo largo de la carrera. Puede concluirse, por lo tanto, que el proyecto se ha llevado a cabo satisfactoriamente,

14.1. Trabajo futuro

Incluso habiendo cumplido con las expectativas del proyecto, hay funcionalidades extras implementables que ampliarían la versatilidad del mismo.

A continuación se destacan las principales mejoras que se podrían implementar en el futuro:

Ampliaciones:

- Posibilidad de uso de otros navegadores importantes para la compañía (Explorer, Safari…):

Habría que investigar el modo en que Selenium2 permite utilizar dichos navegadores, y en caso de integrarse, desarrollar las líneas de código necesarias. Después habría que añadir las opciones en el fichero de properties.

- Usar mockups para la ejecución de los tests

Puesto que los tests se llevan a cabo sobre el portal web directamente, cuando se prueban funcionalidades como el registro de usuarios o la inserción de comentarios, la aplicación los lleva a cabo y persisten en la BBDD. Esto se convierte en un problema cuando el portal es el de producción. Por lo que un modo de hacerlo sería con mocks; se trata de una especie de técnica caja negra en la que se le pasan a las funciones los datos que esperan recibir, pero que evita que las salidas vayan contra el portal web. De este modo se comprueba el correcto funcionamiento de las funcionalidades pero sin afectar a la BBDD.

- Integración con TestLink

Sería necesario bajar la versión de Behat a la 2.5 con todos los cambios correspondientes o esperar a que en la versión actual desarrollen la funcionalidad de poder especificar el tipo de fichero de salida que se desea. En el segundo caso simplemente habría que redirigir la salida al fichero del que TestLink cogiese los datos, configurar TestLink y ejecutarlo.

Mejoras:

- Optimizar la ejecución de todos los tests a la vez.

Actualmente no se puede por la limitación de Selenium2 del número de pestañas totales que permite tener abiertas a la vez. Para ello habría que abrir y cerrar las sesiones con Mink una vez finalizado el test y hechas las capturas de pantalla correspondientes.

14.2. Valoración personal

La oportunidad de haber realizado este proyecto ha permitido contrastar los conocimientos adquiridos en la especialidad de Sistemas de la Información con el mundo real., lo que ha permitido entender, ampliar y mejorar dichos conocimientos. El desarrollo de la herramienta ha posibilitado poner en práctica los conceptos aprendidos a lo largo de la carrera y también descubrir las nuevas tecnologías y metodologías de testeo.

Gracias a haberlo hecho con el convenio tipo B (empresa) se ha tenido la enriquecedora experiencia del trabajo con profesionales del sector, con los que se han intercambiado opiniones muy diversas y de quienes se ha aprendido el modo en el que se lleva a cabo el desarrollo de software en el mundo empresarial y el verdadero sentido del trabajo en equipo.

Pese a haber finalizado el proyecto con éxito, a nivel personal se ha tenido que invertir mucho tiempo en aprender los temas de los que se tenía un conocimiento bastante limitado, como el desarrollo de portales con Symfony, el lenguaje de programación PHP, las metodologías TDD y BDD, la configuración del entorno de trabajo para poder tele-trabajar, etc. Puesto que se trata de un proyecto innovador, en la empresa no se tenía el conocimiento de las metodologías utilizadas, por lo que ha habido retos de bastante nivel a los que ha habido que enfrentarse en solitario, teniendo siempre la presión del limitado margen temporal.

La sensación final es de satisfacción y de gratitud, puesto que se ha aportado y aprendido de cara a la empresa, se han superado retos personales, se ha desarrollado el proyecto con un rango de sostenibilidad considerable (23:30), se ha ampliado la experiencia en el mundo laboral y se ha planificado y gestionado el tiempo correctamente. Todos estos factores han ayudado a crecer en cuanto a responsabilidad y madurez laboral.

15. Anexo

15.1. Diagrama de Gantt inicial del proyecto

	Nombre	Duración	% Completado	Inicio	Fin	Predecesoras	Recursos
1	**Análisis previo**	20d?	0%	**18/01/2016**	**12/02/2016**		**Team Leader**
2	Análisis de sistemas y metodologías actuales	5d?	0%	18/01/2016	22/01/2016		Team Leader
3	Detección de necesidades	5d?	0%	25/01/2016	29/01/2016	2	Team Leader
4	Análisis de oportunidades	5d?	0%	01/02/2016	05/02/2016	3	Team Leader
5	Definición conceptual del proyecto	5d?	0%	08/02/2016	12/02/2016	4	Team Leader
6	**Gestión de proyectos**	32d?	0%	**18/02/2016**	**01/04/2016**	**1**	**Team Leader**
7	Definición del alcance	10d?	0%	18/02/2016	02/03/2016		Team Leader
8	Planificación temporal	5d?	0%	02/03/2016	08/03/2016	7	Team Leader
9	Gestión económica y sostenibilidad	5d?	0%	09/03/2016	15/03/2016	8	Team Leader
10	Presentación preliminar	3d?	0%	16/03/2016	18/03/2016	9	Team Leader
11	Pliego de condiciones de la especialidad de SI	5d?	0%	21/03/2016	25/03/2016	10	Team Leader
12	Presentación y documento final	10d?	0%	21/03/2016	01/04/2016	10	Team Leader
13	**Análisis de Requisitos**	2d?	0%	**07/03/2016**	**08/03/2016**	**1**	**Analista**
14	Descripción detallada de las funcionalidades de la herramient	2d?	0%	07/03/2016	08/03/2016		Analista

15	**Especificación**	4d?	0%	**10/03/2016**	**15/03/2016**	**13**	**Analista**
16	Especificación de Requisitos	2d?	0%	10/03/2016	11/03/2016		Analista
17	Especificación de Diseño	2d?	0%	11/03/2016	14/03/2016	16	Analista
18	Especificación de Módulos	2d?	0%	14/03/2016	15/03/2016	17	Analista
19	**Diseño, Implementación y Pruebas**	96d?	0%	**15/02/2016**	**27/06/2016**	**1**	**Desarrollador**
20	Investigación de portales	20d?	0%	15/02/2016	11/03/2016		Desarrollador
21	Creación portal dummy	10d?	0%	14/03/2016	25/03/2016	20	Desarrollador
22	Creación juego de pruebas dummy	10d?	0%	28/03/2016	08/04/2016	21	Desarrollador
23	Investigación de lenguajes y herramientas digitalización	20d?	0%	11/04/2016	06/05/2016	20	Desarrollador
24	Digitalizar resultados con Selenium/Mink	10d?	0%	09/05/2016	20/05/2016	23	Desarrollador
25	Investigar API's TestLink	5d?	0%	23/05/2016	27/05/2016	20	Desarrollador
26	Integrar los informes obtenidos con TestLink (opcional)	10d?	0%	30/05/2016	10/06/2016	25	Desarrollador
27	Ejecutar los casos de prueba	5d?	0%	13/06/2016	17/06/2016	26	Desarrollador
28	Documentar el proyecto	96d?	0%	23/02/2016	27/06/2016		Desarrollador
29	Seguimiento de la planificación y ejecución del proyecto	96d?	0%	15/02/2016	27/06/2016		Desarrollador[80%],Ponente[20%
30	**Finalización**	5d?	0%	**28/06/2016**	**04/07/2016**	**19**	**Team Leader**
31	Cerrar proyecto	5d?	0%	28/06/2016	04/07/2016		Team Leader

Figura 1: Diagrama de Gantt de la planificación inicial

15.1.1. Leyenda diagrama de Gantt inicial del proyecto

	Team Leader del proyecto
	TFG: Rol de *Team Leader*
	TFG: Rol de Analista
	TFG: Rol de Desarrollador
	TFG: Rol de Desarrollador en paralelo
	TFG: Rol de Desarrollador + Ponente en paralelo

15.2. Diagrama de Gantt inicial del TFG

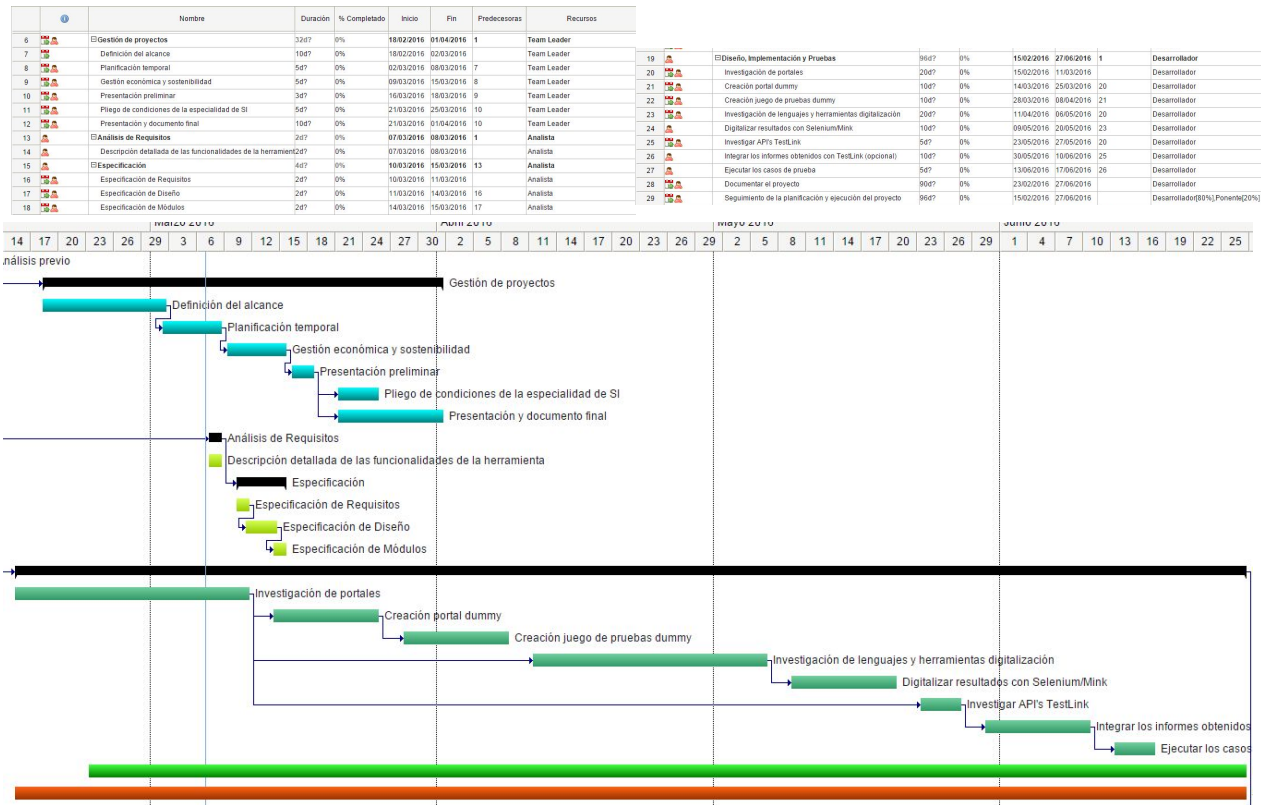
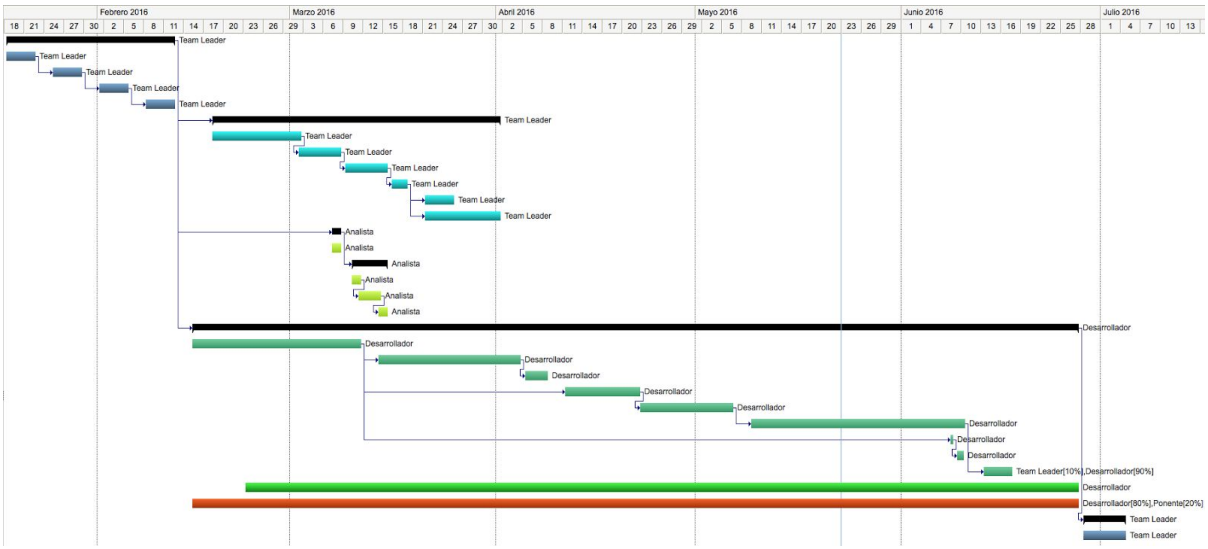


Figura 2: Diagrama de Gantt de la planificación inicial

15.2.1. Leyenda diagrama de Gantt inicial del TFG

	TFG: Rol de *Team Leader*
	TFG: Rol de Analista
	TFG: Rol de Desarrollador
	TFG: Rol de Desarrollador en paralelo
	TFG: Rol de Desarrollador + Ponente en paralelo

15.3. Diagrama de Gantt final del proyecto



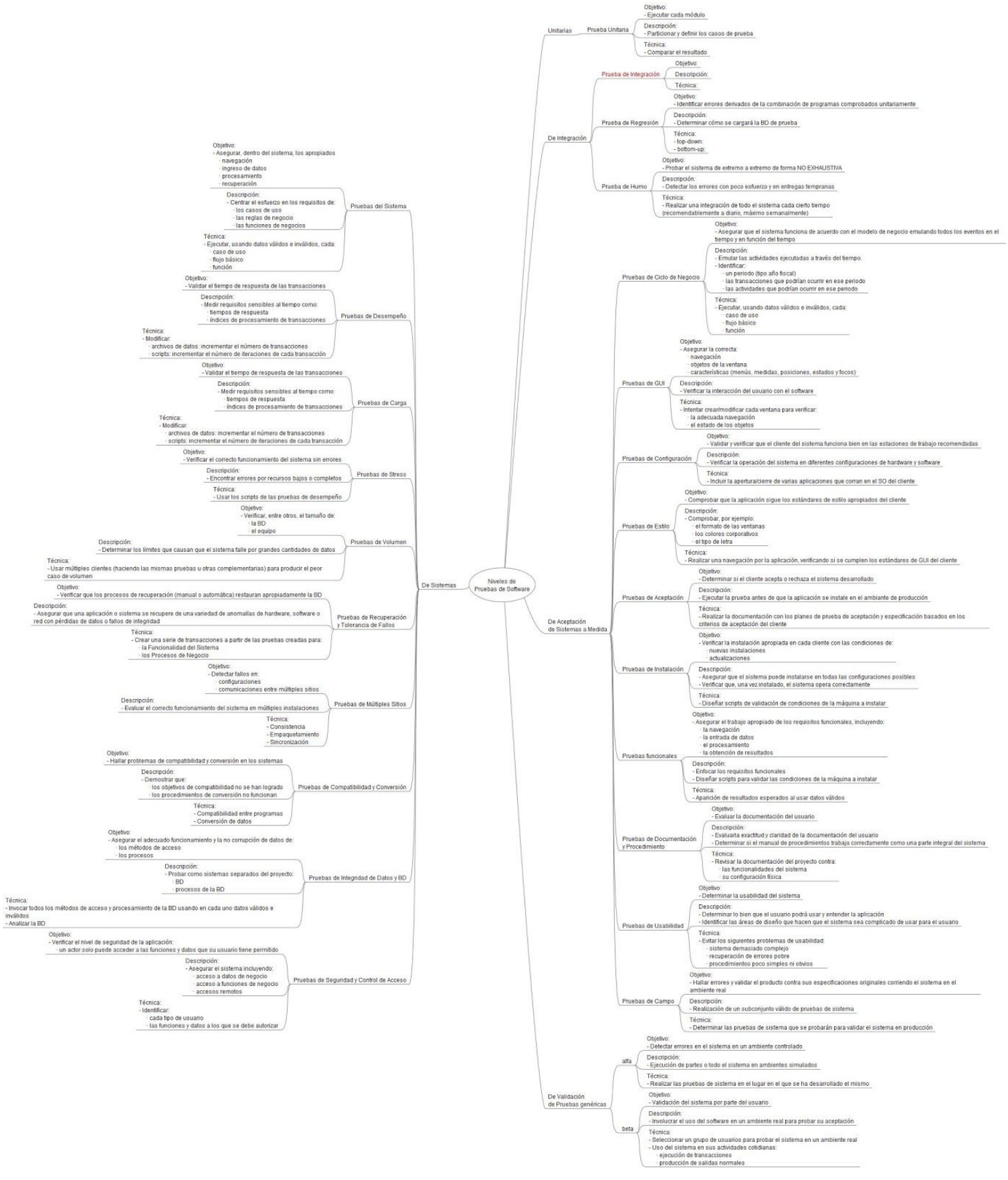
		Nombre	Duración	Inicio	Fin	Predecesoras	Recursos
1		**Análisis previo**	20d?	**18/01/2016**	**12/02/2016**		**Team Leader**
6		**Gestión de proyectos**	32d?	**18/02/2016**	**01/04/2016**	**1**	**Team Leader**
7		Definición del alcance	10d?	18/02/2016	02/03/2016		Team Leader
8		Planificación temporal	5d?	02/03/2016	08/03/2016	7	Team Leader
9		Gestión económica y sostenibilidad	5d?	09/03/2016	15/03/2016	8	Team Leader
10		Presentación preliminar	3d?	16/03/2016	18/03/2016	9	Team Leader
11		Pliego de condiciones de la especialidad de SI	5d?	21/03/2016	25/03/2016	10	Team Leader
12		Presentación y documento final	10d?	21/03/2016	01/04/2016	10	Team Leader
13		**Análisis de Requisitos**	2d?	**07/03/2016**	**08/03/2016**	**1**	**Analista**
14		Descripción detallada de las funcionalidades de la herramienta	2d?	07/03/2016	08/03/2016		Analista
15		**Especificación**	4d?	**10/03/2016**	**15/03/2016**	**13**	**Analista**
16		Especificación de Requisitos	2d?	10/03/2016	11/03/2016		Analista
17		Especificación de Diseño	2d?	11/03/2016	14/03/2016	16	Analista
18		Especificación de Módulos	2d?	14/03/2016	15/03/2016	17	Analista
19		**Diseño, Implementación y Pruebas**	96d?	**15/02/2016**	**27/06/2016**	**1**	**Desarrollador**
20		Investigación de portales	20d?	15/02/2016	11/03/2016		Desarrollador
21		Creación portal dummy	16d?	14/03/2016	04/04/2016	20	Desarrollador
22		Creación juego de pruebas dummy	4d?	05/04/2016	08/04/2016	21	Desarrollador
23		Investigación de lenguajes y herramientas digitalización	9.88d?	11/04/2016	22/04/2016	20	Desarrollador
24		Digitalizar resultados con Selenium/Mink (1er ciclo)	10.13d?	22/04/2016	06/05/2016	23	Desarrollador
25		Digitalizar resultados con Selenium/Mink (resto ciclos)	24.5d?	09/05/2016	10/06/2016	24	Desarrollador
26		Investigar API's TestLink	1d?	08/06/2016	08/06/2016	20	Desarrollador
27		Integrar los informes obtenidos con TestLink (opcional)	1.13d?	09/06/2016	10/06/2016	26	Desarrollador
28		Crear Demo	4.88d?	13/06/2016	17/06/2016	25	Team Leader[10%],Desarrollador[90%]
29		Documentar el proyecto	90d?	23/02/2016	27/06/2016		Desarrollador
30		Seguimiento de la planificación y ejecución del proyecto	96d?	15/02/2016	27/06/2016		Desarrollador[80%],Ponente[20%]
31		**Finalización**	5d?	**28/06/2016**	**04/07/2016**	**19**	**Team Leader**
32		Cerrar proyecto	5d?	28/06/2016	04/07/2016		Team Leader

Figura 3: Diagrama de Gantt de la planificación definitiva

15.3.1. Leyenda diagrama de Gantt final del proyecto

	*Team Leader* del proyecto
	TFG: Rol de *Team Leader*
	TFG: Rol de Analista
	TFG: Rol de Desarrollador
	TFG: Rol de Desarrollador en paralelo
	TFG: Rol de Desarrollador + Ponente en paralelo

15.4. Figura 2: Esquema de los Niveles de Pruebas de Software



16. Referencias

16.1. Bibliografía

[Bec02] Beck, K. (2002). *Test-Driven Development By Example*. [online] Electrical Engineering and Computer Science (EECS). Available at: https://www.google.es/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjDoYamwafNAhUEwBQKHd7uAGMQFgghMAA&url=http%3A%2F%2Fwww.eecs.yorku.ca%2Fcourse_archive%2F2003-04%2FW%2F3311%2FsectionM%2Fcase_studies%2Fmoney%2FKentBeck_TDD_byexample.pdf&usg=AFQjCNFH3sHVtipi1FJBka_-fVg3LCIrVQ&sig2=Xevj0PBtsohkVxKxQVBt8w.

[DPR12] "Documento de Pruebas (DPR)", Gobierno de Cantabria, 2012. https://amap.cantabria.es/amap/bin/view/kitbienvenida/WebHome

[SLO16] Symfony 2.4, e. (2016). *Symfony 2.4, el libro oficial*. [online] Librosweb.es. Available at: https://librosweb.es/libro/symfony_2_x/.

[Som05] Sommerville I. "Ingenieria del Softwarem, Séptima edición", pp 402, 2005.

[Wen01] Wenger E., "Comunidades de práctica: Aprendizaje, significado e identidad", 2001.

16.2. Proyectos

[CV15] Castro Villena, D. (2015). Sistema de control de medicación. *Universitat Politècnica De Catalunya*. Retrieved from http://upcommons.upc.edu/handle/2099.1/25125

[Fer10] Fernández M., "Gestión de un proyecto de control de pruebas de un aplicativo", 2010. https://upcommons.upc.edu/handle/2099.1/10975

[Fue03] Fuentes, J. (2003). *Realidad virtual aplicada al tratamiento del trastorno de lateralidad y ubicación espacial*. Licenciatura. Universidad de las Américas Puebla, Escuela de Ingeniería, Departamento de Ingeniería en Sistemas Computacionales.

[Gar09] Garcia D., "Desarrollo de un sistema para la gestión de logs de seguridad", 2009. https://upcommons.upc.edu/handle/2099.1/7442

[Mas16] Massa C., "Autogeneración de documentación web a partir de un modelo funcional en la herramienta Recover", 2016. https://upcommons.upc.edu/handle/2117/82469

[Men11] Mendoza S., "Estudio de la metodología de automatización del testeo en aplicaciones web para e-Catalunya", 2011. https://upcommons.upc.edu/handle/2099.1/12515

[Per06] Pérez B., "Proceso de testing funcional independiente", 2006. http://www.ces.com.uy/documentos/imasd/Tesis-Beatriz_Perez_2006.pdf

[Tor12] Tort Pugibet, A. (2012). Testing and test-driven development of conceptual schemas. *Universitat Politècnica de Catalunya*. [online] Available at: http://upcommons.upc.edu/handle/10803/83276 [Accessed 2 Jun. 2016].

[Val13] Valderrama D., "Plataforma de validación de una plataforma SDK", 2013.

https://upcommons.upc.edu/handle/2099.1/18581

16.3. Artículos

[Ath12] Athiyarath S., "Integrate Testlink with Jira", FCOOS, 23 Diciembre 2012. http://blogs.fcoos.net/integrate-testlink-with-jira/

[Bal12] Bala, "Selenium and TestLink Integration", Software testing, 5 Marzo 2012. http://balasoftwaretesting.blogspot.com.es/2012/03/selenium-and-testlink-integration.html

[Bec03] Beck, K. (2003). *Test-driven development*. Boston: Addison-Wesley.

[Env15] Envato, (2015). *BDD method life cycle*. [image] Available at: http://webuild.envato.com/blog/making-the-most-of-bdd-part-1/.

[Far11] Farías M., "Automatizamos las pruebas funcionales", 4 Marzo 2011 http://blog.ces.com.uy/?p=142

[Gen09] Gencat, "European eGovernment Awards", 2009 http://www.gencat.cat/especial/eGovernmentAwards09/cat/ecatalunya.htm

[Her16] Hernández, R. (2016). *BDD, Cucumber y Gherkin. Desarrollo dirigido por comportamiento*. [online] Genbetadev.com. Available at: http://www.genbetadev.com/metodologias-de-programacion/bdd-cucumber-y-gherkin-desarrollo-dirigido-por-comportamiento

[Ima09] IMAGINEA, "Setting up TestLink on Linux", 22 Febrero 2009. http://blog.imaginea.com/setting-up-testlink-on-linux/

[Kin15] Kinishita B., Shcherbakov, "TestLink Plugin", 14 Noviembre 2015. https://wiki.jenkins-ci.org/display/JENKINS/TestLink+Plugin

[KF10] Kinoshita B., Fernandes C., "Jenkins TestLink Plug-in Tutorial", 2010. https://wiki.jenkins-ci.org/download/attachments/753702/jenkins.pdf

[Lov14] Lovell G., "Why PHP doesn't suck anymore", VEHIKL, 13 Junio 2014. http://transmission.vehikl.com/why-php-doesnt-suck-anymore/

[Man10] Mancardi F., "How to setup bugtracking system integration Mantis and Bugzilla examples", 2010. http://svn.sourceforge.jp/svnroot/testlinkjp/trunk/docs/tl-bts-howto.pdf

[Nav14] Navarro M., "Integrating Jira with TestLink", SANTEX, 23 Mayo 2014. http://santexgroup.com/blog/integrating-jira-with-testlink-3/

[Nor06] North, D. (2006). Behavior Modification, The evolution of behavior-driven development. *Stickyminds*, 2006(03).

[Nor07] Dan North & Associates. (2007). *What's in a Story?*. [online] Available at: https://dannorth.net/whats-in-a-story/ [Accessed 4 Jun. 2016].

[RTA13] RoadToAutomation (RTA),"Road to integration of WebDriver test with TestLink", Septiembre 2013. http://roadtoautomation.blogspot.com.es/2013/09/road-to-integration-of-webdriver-test.html

[Sti06] StickyMinds. (2006). *StickyMinds | Behavior Modification | Page 1*. [online] Available at: https://www.stickyminds.com/better-software-magazine/behavior-modification.

[TS13] TesteandoSoftware (TS), "Las mejores herramientas para realizar pruebas de software", Mayo 2013. http://testeandosoftware.com/las-mejores-herramientas-para-realizar-pruebas-de-software/

16.4. Presentaciones

[Ail12] Aillon P., "Pruebas de aplicaciones web", 2011. http://es.slideshare.net/paulinaaillon/pruebas-de-aplicaciones-web

[Gut] Gutiérrez, J. (n.d.). *Introducción al Proceso de Pruebas*. [online] Universidad de Sevilla. Available at: http://www.lsi.us.es/~javierj/cursos_ficheros/02.SR.pdf.

[LeG12] Lemus, G. (2012). *Tipos de pruebas de software*. [online] Slideshare. Available at: http://es.slideshare.net/GuillermoLemus/tipos-de-pruebas-de-software.

16.5. Páginas web oficiales

[AbJ] Abad, J. (n.d.). [online] LinkedIn. Available at: https://co.linkedin.com/in/jorgeabadl.

[AbJ05] Abad, J. (2005). *Ingeniería de Software: TIPOS DE PRUEBAS DE SOFTWARE*. [online] Ing-sw.blogspot.com.es. Available at: http://ing-sw.blogspot.com.es/2005/04/tipos-de-pruebas-de-software.html.

[BAC16] Cucumber.io. (2016). *Browser Automation·Cucumber*. [online] Available at: https://cucumber.io/docs/reference/browser-automation [Accessed 6 Jun. 2016].

[BaJ] Bach, J. (n.d.). *Satisfice, Inc.*. [online] Satisfice.com. Available at: http://satisfice.com.

[Beh] Docs.behat.org. (n.d.). *Behat Documentation — Behat 3.0.12 documentation*. [online] Available at: http://docs.behat.org/en/v3.0/.

[Beh10] Docs.behat.org,. (2010). *Behat Documentation — Behat 3.0.12 documentation*. Retrieved from http://docs.behat.org/

[CDT01] Context-driven-testing.com. (n.d.). [online] Available at: http://context-driven-testing.com.

[CES08] Ces.com.uy,. (2010). *CES — Centro de Ensayos de Software*. Retrieved from http://www.ces.com.uy/

[CFP16] Symfony.com. (2016). *Create your First Page in Symfony (The Symfony Book)*. [online] Available at: http://symfony.com/doc/current/book/page_creation.html.

[ChD] Sites.google.com. (n.d.). *Getting started - ChromeDriver - WebDriver for Chrome*. [online] Available at: https://sites.google.com/a/chromium.org/chromedriver/getting-started.

[Chr] Google.com. (n.d.). *Chrome Browser*. [online] Available at: https://www.google.com/chrome/browser/desktop/index.html.

[Chu] Chromium.org. (n.d.). *The Chromium Projects*. [online] Available at: https://www.chromium.org.

[Com12] Getcomposer.org,. (2012). *Composer*. Retrieved from https://getcomposer.org/

[Com16] Getcomposer.org. (2016). *Composer*. [online] Available at: https://getcomposer.org.

[DB16] Docs.behat.org. (2016). *Configuration - behat.yml — Behat 2.5.3 documentation*. [online] Available at: http://docs.behat.org/en/v2.5/guides/7.config.html [Accessed 7 Jun. 2016].

[DOC] Doctrine-project.org. (n.d.). *About — Doctrine Project*. [online] Available at: http://www.doctrine-project.org/about.html.

[Esi16] Esios.ree.es. (2016). *Bienvenido | ESIOS electricidad · datos · transparencia*. [online] Available at: https://www.esios.ree.es/es.

[Eve96] Everis.com,. (1996). *everis*. Retrieved from http://www.everis.com

[Fir] Mozilla. (n.d.). *Firefox*. [online] Available at: https://www.mozilla.org/es-ES/firefox/new/.

[FL05] Funkload.nuxeo.org,. (2005). *FunkLoad 1.17.1 documentation*. Retrieved from http://funkload.nuxeo.org/intro.html

[FOS] Knpbundles.com. (n.d.). *FOSUserBundle by FriendsOfSymfony | KnpBundles*. [online] Available at: http://knpbundles.com/FriendsOfSymfony/FOSUserBundle.

[Gal08] Garcerant, I. (2008). *Pruebas de Software | Tecnología y Synergix*. [online] Synergix. Available at: https://synergix.wordpress.com/tag/pruebas-de-software/.

[Gan09] Gantter.com, (2009). *Gantter - web-based project scheduling made easy*. [online] Available at: http://gantter.com/.

[GD12] Google.com, (2012). *Google Drive: Emmagatzematge al núvol i còpia de seguretat de fotos, documents i molt més*. [online] Available at: https://www.google.com/intl/ca/drive/.

[GH08] GitHub, (2008). *Build software better, together*. [online] Available at: https://github.com/.

[GH10] GitHub, (2010). *Testcase Management Database and Front-end*. [online] Available at: https://github.com/viglesiasce/testlink/ [Accessed 3 Mar. 2016].

[GHC16] GitHub. (2016). *cucumber/gherkin*. [online] Available at: https://github.com/cucumber/gherkin/blob/master/gherkin-languages.json [Accessed 6 Jun. 2016].

[Ghe] GitHub. (n.d.). *Cucumber/Gherkin*. [online] Available at: https://github.com/cucumber/cucumber/wiki/Gherkin.

[GM04] Mail.google.com, (2004). *Gmail*. [online] Available at: https://mail.google.com/.

[Gnu84] Gnu.org,. (1984). *The GNU Operating System and the Free Software Movement*. Retrieved from http://www.gnu.org

[Gou16] GitHub. (2016). *FriendsOfPHP/Goutte*. [online] Available at: https://github.com/FriendsOfPHP/Goutte [Accessed 8 Jun. 2016].

[ICS16] Symfony.com. (2016). *Installing and Configuring Symfony (The Symfony Book)*. [online] Available at: http://symfony.com/doc/current/book/installation.html.

[KaC] Kaner, C. (n.d.). *Cem Kaner, J.D., Ph.D.*. [online] Kaner.com. Available at: http://kaner.com.

[Lif00] Liferay.com,. (2000). *Soluciones para Portal y Colaboración Social Open Source. - Liferay.com*. Retrieved from https://www.liferay.com/es/

[MaB] Marick, B. (2016). *Exampler Consulting*. [online] Exampler.com. Available at: http://www.exampler.com.

[MCB16] Mink.behat.org. (2016). *Controlling the Browser — Mink 1.6 documentation*. [online] Available at: http://mink.behat.org/en/latest/guides/session.html [Accessed 8 Jun. 2016].

[Min11] Mink.behat.org,. (2011). *Welcome to the Mink documentation! — Mink 1.6 documentation*. Retrieved from http://mink.behat.org/

[Min16] Mink.behat.org. (2016). *Welcome to the Mink documentation! — Mink 1.6 documentation*. [online] Available at: http://mink.behat.org/en/latest/ [Accessed 7 Jun. 2016].

[MIP16] Mink.behat.org. (2016). *Interacting with Pages — Mink 1.6 documentation*. [online] Available at: http://mink.behat.org/en/latest/guides/interacting-with-pages.html [Accessed 8 Jun. 2016].

[MMP16] Mink.behat.org. (2016). *Manipulating Pages — Mink 1.6 documentation*. [online] Available at: http://mink.behat.org/en/latest/guides/manipulating-pages.html [Accessed 8 Jun. 2016].

[MTP16] Mink.behat.org. (2016). *Traversing Pages — Mink 1.6 documentation*. [online] Available at: http://mink.behat.org/en/latest/guides/traversing-pages.html [Accessed 8 Jun. 2016].

[OST03] Opensourcetesting.org,. (2003). *Open source software testing tools*. Retrieved from http://www.opensourcetesting.org/

[OWS14] Oracle.com,. (2014). *Oracle WebCenter Sites | Oracle Technology Network*. Retrieved from http://www.oracle.com/technetwork/middleware/webcenter/sites/overview/index.html

[PC] Buenas Tareas. (2011). *Pruebas de configuracion - Ensayos universitarios - Mondrito*. [online] Available at: http://www.buenastareas.com/ensayos/Pruebas-De-Configuracion/1645724.html.

[PeB] Pettichord, B. (n.d.). *Bret Pettichord*. [online] Pettichord.com. Available at: http://pettichord.com.

[PDA] Pruebasdesoftware.com. (2005). *La prueba de aceptación*. [online] Available at: http://www.pruebasdesoftware.com/pruebadeaceptacion.htm.

[PHP95] Php.net,. (1995). *PHP: Hypertext Preprocessor*. Retrieved from http://php.net/

[Pig15] Pigeon, X. (2015). *Lifecycle of the Test-Driven Development method*. [image] Available at: https://en.wikipedia.org/wiki/File:TDD_Global_Lifecycle.png [Accessed 2 Jun. 2016].

[PP16] Page Personnel. (2016). *Estudios de Remuneración | Page Personnel*. [online] Available at: http://www.pagepersonnel.es/prensa-estudios/estudios-remuneracion [Accessed 14 Mar. 2016].

[PUS16] Symfony.com. (2016). *Projects using Symfony*. [online] Available at: http://symfony.com/projects.

[ReC16] Cucumber.io. (2016). *Reference-Cucumber*. [online] Available at: https://cucumber.io/docs/reference [Accessed 6 Jun. 2016].

[Sah16] Sahipro.com. (2016). *Introduction*. [online] Available at: http://sahipro.com/docs/introduction/index.html [Accessed 8 Jun. 2016].

[Sel04] Seleniumhq.org,. (2004). *Selenium - Web Browser Automation*. Retrieved from http://www.seleniumhq.org

[Sel16] Docs.seleniumhq.org. (2016). *Selenium - Web Browser Automation*. [online] Available at: http://docs.seleniumhq.org [Accessed 8 Jun. 2016].

[Sel2] Mink.behat.org. (n.d.). *Selenium2Driver — Mink 1.6 documentation*. [online] Available at: http://mink.behat.org/en/latest/drivers/selenium2.html.

[SGR16] Symfony.com. (2016). *6 good reasons to use Symfony*. [online] Available at: http://symfony.com/six-good-reasons.

[Sha13] Products.office.com,. (2013). *SharePoint: Herramientas de software de colaboración en grupo*. Retrieved from https://products.office.com/es-es/sharepoint/collaboration

[Spe] *SpecFlow*. *SpecFlow*. Retrieved from http://www.specflow.org

[Sta16] Stackoverflow.com. (2016). *Stack Overflow*. [online] Available at: http://stackoverflow.com.

[STE] Symfony.com. (n.d.). *Symfony Twig Extensions (The Symfony Reference)*. [online] Available at: http://symfony.com/doc/current/reference/twig_reference.html.

[SW] Seleniumhq.org. (n.d.). *Selenium WebDriver*. [online] Available at: http://www.seleniumhq.org/projects/webdriver/.

[Sym05] Symfony.com,. (2005). *Symfony, High Performance PHP Framework for Web Development*. Retrieved from https://symfony.com/

[TL13] Testlink.org,. (2013). *TestLink*. Retrieved from http://testlink.org

[TRP16] Symfony.com. (2016). *The Release Process (Contributing to Symfony)*. [online] Available at: http://symfony.com/doc/current/contributing/community/releases.html.

[UB16] Ub.edu.ar, (2016). *Especificaciones de Software*. [online] Available at: http://www.ub.edu.ar/catedras/ingenieria/ing_software/ubftecwwwdfd/espsoft/espsoft.htm

[UCM16] Universidad Complutense de Madrid,. (2016). *FATIGA LABORAL: CONCEPTOS Y PREVENCIÓN*. Retrieved from http://www.ucm.es/data/cont/docs/3-2013-02-18-1-FATIGA%20LABORAL.%20CONCEPTOS%20Y%20PREVENCIÓN.pdf

[Zyn08] Zyncro,. (2008). *Your Enterprise Social Network - Zyncro*. Retrieved from http://www.zyncro.com/